Technical Report Series

TRW

PSEE Architecture Report

Architectures and Models for Next Generation Process-based Software Engineering Environments

PSEE Architecture Report

Architectures and Models

for Next Generation Process-based

Software Engineering Environments

TRW Systems Integration Group
Redondo Beach, California 90278

Sponsored by

Advanced Research Projects Agency (ARPA)
and Space and Naval Warfare Systems Command
ARPA Order No. B343
Under SPAWAR Contract # N00039-95-C-0017

February 1995

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1995	Interim

4. TITLE AND SUBTITLE

PSEE Architecture Report
PSEE Architecture Report Attachment - Section 11

5. FUNDING NUMBERS

SPAWAR C#
N00039-95-C-0017
ARPA Order #B343

6. AUTHOR(S)

Maria H. Penedo (author, editor)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

TRW
One Space Park
Redondo Beach, CA 90278

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

SPAWAR, 2451 Crystal Drive, Arlington, VA 22245-54200
ARPA, 3701 North Fairfax Drive, Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution Statement A: Approved for public release, distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This Architecture Report documents investigations towards the definition of a Process-based Software Engineering Environment (PSEE) Reference Architecture. Those investigations are in support of the definition of a component-based architectural approach for the rapid construction of PSEEs. It represents work in progress.

These investigations were conducted within the "Architectures and Models for Next Generation Process-based SEEs" project; they also revise and enhance prior related work. This document is to be the first in a series of Architecture Reports to be delivered by this contract (pending continuous funding).

This document actually consists of eleven (11) reports and four (4) references which provide technical data related to the current state (art and practice) of SEEs including architectural recommendations, requirements, and lessons learned; the data was gathered from national and international efforts and systems, and includes recent developments in the commercial sector and research programs. A key underlying assumption of this work is community participation and community consensus; therefore, some of the documents have been jointly developed with members of the community.

14. SUBJECT TERMS	15. NUMBER OF PAGES
architectures, process, software engineering environment	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

Outline

0. Introduction to the report.

1. Lessons Learned in Designing and Implementing Life-Cycle Generic Models, by M. Penedo.

2. Workshop Overview, by Narayanaswamy, K., et al.

3. Report on the 1989 Software CAD Databases Workshop., by L Rowe.

4. Life-cycle (Sub) Process Scenario, by M. Penedo.

5. ISPW9 Process Demonstrations - Summary, by M. Penedo.

6. SBUS: A Framework for Software Bus Comparison, by M. Penedo.

7. SEE Software Bus Survey, by C. Shu and M. Penedo.

8. ARPA Interoperability Matrix, by D. Heimbigner.

9. ARPA Interoperability Working Group - Summary Charts, by D. Heimbigner and others.

10. Architecture Bibliography.

11. PSEE Tutorial: Trends in the Construction of Next Generation Software Engineering Environments, by M. Penedo.

1 Introduction

This Architecture Report documents investigations towards the definition of a Process-based Software Engineering Environment (PSEE) Reference Architecture. Those investigations are in support of the definition of a component-based architectural approach for the rapid construction of PSEEs. It represents work in progress.

These investigations were conducted within the *Architectures and Models for Next Generation Process-based SEEs* project; they also revise and enhance prior related work. These investigations took into consideration recent developments in the commercial sector and research programs. This document is to be the first in a series of Architecture Reports to be delivered by this contract (pending continuous funding).

This document actually consists of eleven (11) reports and four (4) references which provide technical data related to the current state (art and practice) of SEEs including architectural recommendations, requirements, and lessons learned; the data was gathered from national and international efforts and systems. A key underlying assumption of this work is community participation and community consensus; therefore, some of the documents have been jointly developed with members of the community.

As part of our community consensus formation activities, in the past few years, we took leadership positions and active participation in various efforts in the community with respect to SEE definition, standardization and assessment. Among the efforts we participated, we include: i) support and participation in the organization of conferences and workshops such as the International Process Workshops, the International Conference on Software Engineering, and the ARPA SEE workshops; ii) past involvement with the NIST Integrated SEE Working Group which defined a SEE Reference Model for SEE Frameworks together with with the European Computer Manufacturer Association (ECMA) TGRM Working Group); iii) maintaining close ties with the major ARPA programs including Arcadia, STARS, Prototech and DSSA communities; iv) serving as liaison between the American and European communities including cooperating with the Eureka Software Factory Project; v) providing support to Government activities/programs such as I-CASE and the Software Technology Support Center (STSC); and vi) keeping track of commercial efforts which are attempting to develop standard products or candidates for consensus formation.

Attached Documents. The documents attached constitute this Technical Report. They document architectural issues including integration, they survey and assess existing PSEE systems, they document initial efforts towards a PSEE reference architecture, and they report collaborative activities in the community. They are of four kinds of authorship, for which we will identify codes and associate with the specific documents:

1. Project Reports (PR), i.e., documents generated solely by project members.
2. Community Reports (CR), documents co-authored or co-edited with members of the international community.

3. Related Reports (RR), i.e., documents (co-)authored by project members on other projects or activities.

4. External Reports (ER), i.e., related documents not co-authored by project members.

We also provide references to important related documents.

Outline

A. Documents related to the Database and Software Engineering Workshop, Sorrento, Italy, May 1994:

1. Lessons Learned in Designing and Implementing Life-Cycle Generic Models, by M. Penedo. **(RR)**

2. Workshop Overview, by Narayanaswamy, K., et al. **(ER)**

3. Report on the 1989 Software CAD Databases Workshop., by L Rowe. **(RR)**

B. Documents related to the International Software Process Workshop, Arlie, October 1994

4. Life-cycle (Sub) Process Scenario, by M. Penedo. **(PR,RR)**

5. ISPW9 Process Demonstrations - Summary, by M. Penedo. **(PR)**

C. Documents related to specific work towards a SEE Reference Architecture.

6. SBUS: A Framework for Software Bus Comparison, by M. Penedo. **(PR)**

7. SEE Software Bus Survey, by C. Shu and M. Penedo. **(RR)**

8. ARPA Interoperability Matrix, by D. Heimbigner. **(ER)**

9. ARPA Interoperability Working Group - Summary Charts, by D. Heimbigner and others. **(CR)**

10. Architecture Bibliography. **(PR)**

11. PSEE Tutorial: Trends in the Construction of Next Generation Software Engineering Environments, by M. Penedo. **(RR,PR)**

D. References to related work towards a reference architecture for SEEs

- *A Survey of Software Engineering Environment Architectural Approaches, by Penedo et al.* **(RR)**
- *NIST/ECMA Reference Model (RM) for SEE Frameworks, NIST Special Publication 500-211, Technical Report ECMA TR/55, 3rd Edition, August 1993.* **(RR)**
- *NGCR Reference Model for Project Support Environments. Brown, A., D. Carney, P. Oberndorf, M. Zelkowitz - editors, NIST Special Publication 500-213, November 1993.* **(ER)**
- Principles of CASE Tool Integration, by A. Brown et al, Oxford University Press, 1994. **(ER)**

Next section provides a summary of those documents.

2 Summary of documents

A. Documents related to the *Database and Software Engineering Workshop*, Sorrento, Italy, May 1994.

They relate to "Object/Data Management" needs in PSEEs.

1. **Lessons Learned in Designing and Implementing Life-Cycle Generic Models, by M. Penedo** (accepted for publication)

   Common data models and common process models are recognized as key integration ingredients in Process-based Software Engineering Environments[1] (PSEE). This paper discusses some lessons learned in designing and implementing such models with emphasis on Object Management (OM) needs. The discussion is based on our experiences derived from prior and current work in process modeling and implementation.

2. **Workshop Overview, by Narayanaswamy, K., et al.**

   This paper was written by the workshop chair and two rapporteurs summarizing the two days of the workshop. There were 32 attendees from 8 countries. Highlights of the workshop included the facts that: there is evidence that considerable amount of work exists towards the development of DBMSs to support software engineering; however, most existing DBMSs still cannot support all of the PSEE requirements. Object oriented database systems appear to provide good support (as our experiences proved a few years ago). A key on-going experiment is the GoodStep project which is using the O2 object-oriented DBMS as the back end for the Merlin process based system and others. Their experience matches our prior findings. We also identified the fact that both communities (SEE and DB) have a much better understanding of each other's needs and capabilities as compared to last workshop, held in Napa Valley, CA, in 1989.

3. **Report on the 1989 Software CAD Databases Workshop, by L Rowe.**

   This was the summary of the first workshop on databases and software engineering, held in Napa Valley, CA, in 1989. At this workshop, specific features identified as needed were: object-oriented data models, navigational and set-oriented query languages, complex object support, long transaction support, derived data support, and alerters.
   (Note: Penedo located this report for distribution to the attendees. Since it was not easily accessible, it is included in this deliverable.)

[1] Process-based environments (PSEE) are environments where both the user interaction paradigm and the execution of its components are process driven.

B. Documents related to the *International Software Process Workshop*, Arlie, October 1994 (both documents will be published in the Proceedings).

4. Life-cycle (Sub) Process Scenario, by M. Penedo.

In the last few years, the process community has defined a "process scenario" which describes a sub-set of the software development process activities, to serve as a canonical example for the community. It was done in conjunction with the International Software Process Workshops (ISPW); Penedo participated in all working groups. In the first years, the scenario was used to understand and compare process modeling notations. In the last two years, this scenario is being demonstrated in existing research and commercial tools and PSEEs; it has served to highlight the difference of existing approaches and to identify the strengths and weaknesses of such approaches. Penedo was the coordinator of the last revision/extension of the process example and the coordinator of the example/demonstration at ISPW9 which was held in October 1994.

This paper contains the ISPW9 scenario or process example. This year's scenario is a revision of the ISPW6 scenario, to take away some of its rigidity, to make it more realistic and tailorable, and to add items related to human-computer interaction and computer mediated human cooperation.

5. ISPW9 Process Demonstrations - Summary, by M. Penedo.

A process demonstration day was held at the 9th International Software Process Workshop (ISPW9), Arlie, VA, October 1994. The objective of the demonstration day was two-fold:

- to evaluate how different systems and environments support/guide users in the fulfillment of their project activities, and
- to bring about technical issues identified by the different implementors in the context of their formalisms and systems.

Eight systems were accepted for demonstration: Oikos, from Pisa University; Synervision, from Hewlet-Packard; Hakoniwa, from Osaka University; LEU, from Lion; MVP-S, from Kaiserslautern University; Oz, from Columbia University; Regatta, from Fujitsu; SPADE, from P. Milano. A scenario example was defined (see document 4) to represent issues in the life of real projects and to serve as a common example for demonstration purposes.

The demonstration day was a success since it provided visual means for understanding and discussing the various PSEE interaction paradigms. Many feel that interspersing demonstrations with the workshop sessions will enrich the discussions and provide more concrete data for discussions. Since the audience consisted of mostly PSEE builders, there was a lot of interest in understanding the architectures of such systems (not obvious during the demonstrations). It is felt that a lot more discussion and understanding is needed about how those systems are constructed, how the architecture of systems support the specific process modeling and enactment

techniques, and what are the relationships among architectures, run-time support for process enactment, user interaction paradigms, and the various characteristics demonstrated. Architecture depictions of the various systems are included in the appendix of this document.

This document gives some background to the scenario, describes the systems demonstrated, and provides a bibliography of related documents.

C. Documents related to specific work towards a SEE Reference Architecture.

These documents represent work in progress dealing with issues of integration and interoperability of PSEE components. This work is being done internally as part of this contract and externally, jointly with D. Heimbigner from University of Colorado and the ARPA SEE Interoperability working group.

6. **SBUS: A Framework for Software Bus Comparison, by M. Penedo,** submitted to ICSE-17 Workshop on Architectures for Software Systems.

   This paper outlines an initial framework, denoted SBUS, for the characterization and comparison of systems or mechanisms which are identified as software buses. "Software Buses" play an important role in supporting component interoperability in Software Engineering Environment (SEE) architectures. The SBUS framework consists of a set of attributes which together characterize such systems; such framework is one of the elements of a PSEE reference architecture. This paper outlines the SBUS attributes and characteristics and illustrates its use by characterizing aspects of the Arcadia's Q system.

   An initial survey based on this framework appears in a technical report (document #7). Both the framework and the survey represent work in progress.

7. **SEE Software Bus Survey, by C. Shu and M. Penedo.**
   This document presents an initial survey of systems which have been characterized as software buses and play an important role in tying together components in Software Engineering Environment (SEE) architectures. The systems surveyed are: HP's BMS, Forest, ESF K/1's Software Bus, ESF Kernel/2r's Muse, Polylith, Arcadia's Q, Weaves. A framework consisting of attributes and characteristics was defined and used for describing those systems' characteristics.

8. **ARPA Interoperability Matrix, by D. Heimbigner.**

   D. Heimbigner and Penedo are currently working on an interoperability framework which will ease the task of evaluating existing interoperability mechanisms. This framework incorporates data from separate studies, Heimbigner's original matrix and Penedo's SBUS model. By describing existing systems using this framework, one should be able to understand the differences among those existing systems. This framework is part of a PSEE reference architecture.

   This paper summarizes Heimbigner's interoperability matrix, as of December 1994. As of today, Penedo and Heimbigner are discussing the merge of both frameworks.

9. **ARPA Interoperability Working Group - Summary Charts, by D. Heimbigner, M. Penedo and others.** There were architecture/interoperability working groups formed at two ARPA SEE meetings in 1994, one on February 14-16 and another one on Sep 21-23. Penedo chaired the first one; at the latter meeting, D. Heimbigner took over the leadership due to the uncertainty of continuous funding for the current contract. The objective of this working group was to identify the current state of the art in this area and to draw a roadmap for ARPA activities. The charts attached summarize the working group discussions.

   During the meetings it was identified that CORBA (Common Object Request Broker Architecture) appears to be the leading mechanism (for the time being) for object and tool interoperability in PSEEs which are UNIX-based. It is believed that CORBA does not support all the needed requirements but the support for CORBA seems to be strong among the software producers and many implementations are starting to appear.

10. **Architecture Bibliography.** This is a list of papers related to PSEE architectures, collected during the course of our investigations.

11. **PSEE Tutorial: Trends in the Construction of Next Generation Software Engineering Environments, by M. Penedo.**
    (Note: due to its length and the fact that copies have already been provided to ARPA and SPAWAR, this tutorial is provided as an attachment)

    This tutorial surveys issues related to Process-based Software Engineering Environments (SEE). It includes definitions and concepts, models, integration characterizations and architectural forms, support for life-cycle process definition and enactment, and SEE reference models. It includes examples of existing SEEs. An informal perspective of software engineering environment architecture evolution is also presented indicating trends and future directions. It also includes cost and productivity highlights. The trends include:

    - In *Architectures:* Client-Server, Distribution, Autonomy, Interoperability, Active, Component-based, Process-based, User-tailorable.
    - In *Processes:* Architecture-driven, Reused-based, Design by Teams, Cooperative (CSCW), Business-driven.

D. References to related work towards a reference architecture for SEEs.

This section contains references to (sometimes jointly co-authored) documents related to community consensus activities in the area of (process-based) SEEs. Those documents are not included here. However, they reflect our active participation in community activities and illustrate the broad spectrum of issues related to software engineering environments. The first document is our survey of commercial and research SEEs and their mapping to a common SEE reference model denoted CEARM; the second document is a joint ECMA/NIST publication, which is now in wide use in the community as a common

basis or model for describing SEE framework functionality; the third document is the PSE Reference Model, which provides a basis for describing SEE User Services. All of these models can be elements in a PSEE Reference Architecture.

- **A Survey of Software Engineering Environment Architectural Approaches, by M. Penedo, A. Karrer and C. Shu, TRW Technical Report IMPSEE-TRW-93-007, November 1993.**

  This document contains the architectural survey which inspired our work in this project. It summarizes our experiences in describing and comparing SEEs using a conceptual model denoted Conceptual Environment Architecture Reference Model (CEARM); it also includes some of our experiences in the development of the NIST/ECMA reference model. The systems surveyed represent recent developments in environment architectures in the commercial sector and research programs.

  The systems investigated as part of the survey were: A Tool Integration Standard (ATIS), Arcadia-1 Architecture, Atherton's Software Backplane, Common APSE Interface Set, Eureka Software Factory Architecture, ESF Kernel/2r, European Advanced Software Technology (EAST) Environment, HP's Softbench, Portable Common Tool Environment, Software Life Cycle Support Environment (SLCSE), SUN's Network Software Environment, Pact Environment. The lessons learned as a result of this survey have also been collected in the document; a summary paper entitled "Towards understanding Software Engineering Environments" has been written.

- **NIST/ECMA Reference Model (RM) for SEE Frameworks.**
  This report is published jointly as an ECMA Technical Report and a NIST Special Publication. *[NIST Special Publication 500-211, Technical Report ECMA TR/55, 3rd Edition, August 1993].*

  Work in a Reference Model (precursors to reference architecture concepts) for Software Engineering Environments (SEE) has been in progress for the past years both in the United States and Europe. A key objective of the work in reference models for SEEs has been to find better ways to describe SEEs and to assist the SEE architectural building process. It is hoped that the concepts described within the reference model can guide the evolution of SEE environment architectures.

  This document describes the RM and was published jointly by the European Computer Manufacturers Association (ECMA) and NIST. This RM is now in wide use in the community as a means of describing and comparing SEEs. The Navy PSESWG[2] effort has also adopted and extended the NIST model to include user functionality (see next document referenced).

  Its authors are the leaders of the NIST/ISEE Working Sub-Groups: M. Penedo (TRW) for Object Management, H. Hart (TRW) for Process Management, T. Oberndorf (NADC) for Interface

[2]Program Support Environment Standards Working Group, part of the Next Generation Computer Resources (NGCR) program.

and Platform, B. Bagwill/M. Zelkowitz (NIST) for User Interface; P. Oberndorf/M. Penedo for Integration; M. Zelkowitz (U. Maryland) was the document editor.

- **NGCR Reference Model for Project Support Environments, by Brown, A., D. Carney, P. Oberndorf, M. Zelkowitz - editors**, Technical Report CMU/SEI-93-TR-23, ESC-TR-93-199, also published as NIST Special Publication 500-213, November 1993.

  The Navy Project Support Environment (PSE) Working Group, part of the Next Generation Computer Resources (NGCR) program, also generated a Reference Model for SEEs. This effort has adopted and extended the NIST model to include user functionality. End-user services are sub-divided into Technical Engineering, Technical Management, Project Management and Support services. This RM does complement the NIST RM by going beyond framework characteristics to include the project users' functional capabilities.

- **Principles of CASE Tool Integration, by A. Brown et al, Oxford University Press, 1994.**

  This is a recent book written by Software Engineering Institute personnel with many interesting lessons learned. The book has the following aims:

  - to assemble existing knowledge on the topic of integration in a CASE environment,
  - to indicate the range of perspectives on the meaning of, and approaches to, integration in a CASE environment,
  - to raise awareness and understanding of the key aspects of CASE environment technology, and the important role that it plays,
  - to showcase SEI work in their CASE Environments project, and
  - to introduce a new model of CASE environment integration that can be used to analyze existing CASE environment, and can provide the basis for constructing a CASE environment with appropriate integration characteristics.

Lessons Learned in Designing and Implementing Life-cycle Generic Models*

Maria H. Penedo
TRW
One Space Park
Redondo Beach, CA 90278

Abstract

Common data models and common process models are recognized as key integration ingredients in Process-based Software Engineering Environments[1] (PSEE). This paper[2] discusses some lessons learned in designing and implementing such models with emphasis on Object Management (OM) needs. The discussion is based on our experiences derived from prior and current work in process modeling and implementation.

Introduction/Background.

There is currently a great deal of activity within the software engineering community in the area of providing automated support for aspects of the software development process. While many successes have been made in individual areas, perhaps the greatest challenge is to *integrate* these successes to produce an effective and integrated automated environment that supports the complete software development life cycle. A unifying (product or process) life-cycle model, which serves as a logical model for the integration of PSEE components, is a key ingredient in support of data integration. Examples of such life-cycle data models are: the Project Master Database Model (PMDB) [PS85] and the SLCSE model [Tay89]. A key distinction between the PMDB model and the SLCSE model is the fact that the first was designed to be generic and the latter supports a more specific family of processes conforming to MIL-STD-2167A.

Data models model data in applications; process models model processes; these models can be implemented in one or more data management systems (DMS). Formalisms (sometimes also called data models) are used to provide representations for data or process models. There are several well known data modeling techniques and formalisms whose merits have been documented in the literature. Recently these techniques are being explored and enhanced for the modeling of software engineering processes. Examples of data formalisms are: relational, entity-relationship-attribute, semantic data model, object-oriented. Examples of formalisms currently used for modeling processes are: state transition models, object-oriented models, rule-based models, Petri-nets, etc. The need for more precise process models and formalisms which are conducive to automation has been discussed in the literature and the search for better models, formalisms and supporting mechanisms continues, as shown by recent published papers, including the ones submitted to the International Workshops on the Software Process.

We have spent many years investigating process modeling and implementation issues, as part of the PMDB work. The original PMDB model [PS85] was expressed in an E-R formalism; it defines a generic life-cycle model supporting the full life-cycle process. It consists of 32 entity types, approximately 200 attributes associated with the various types, and 200 relationships between the various types. It concentrated on the life-cycle data and relationships even though it did include aspects of the process embedded in it, as illustrated by the entities Accountable Task (model-

*In Proceedings of the Database and Software Engineering Workshop, Sorrento, Italy, May 1994

[1] Process-based environments (PSEE) are environments where both the user interaction paradigm and the execution of its components are process driven.

[2] This paper contains revised excerpts of text from [PS91].

ing development and management tasks), Milestones (modeling certain project events) and Person (modeling process resources). The enhanced PMDB model, denoted PMDB+ model [PS91], extended a subset of the original PMDB to include explicit process behavior. It uses an "extended E-R model" as its formalism; the extension includes operations and conditional events.

A large part of our latest investigations concentrated on gaining experience with executable process models with the objectives of: determining the impact of process encoding on SEE components such as object management services, user interface management services and existing tools. We feel that our experience with alternative modeling and implementation approaches provided us with valuable insight into many of the issues and solutions. Some of our experiences with respect to the object management support for these models and their implementation are described here. The full PMDB model can be found in [PS84] and initial lessons learned in [PS85]. More details of the lessons learned with the enhanced model can be found in [PS91]; and lessons learned as they apply to evolution can be found in [Pen93].

Lessons Learned

Our first prototyping exercise implemented a subset of the PMDB model in a relational data management system. The objectives of this exercise were to assess the validity of the PMDB model and to investigate relevant issues associated with the automation of such process models. Those investigations identified weaknesses in both the relational and ER approaches and identified representation requirements for environment support components with special emphasis on object management (OM), including the need for abstract interfaces, strong typing, computed attributes, conditionally triggered procedures, integration of textual (e.g., files) and relational data, process-based user interfaces, automated mapping support, consistency and inconsistency management. A summary of conclusions and observations which resulted from this exercise is provided in [Pen86].

A few lessons learned from designing life-cycle data models include:

- It can serve as precise data descriptions for tool interoperability.
- It can serve as the user's cognitive model for environment interaction.
- The model and its implementation should be separate but mappable to each other.
- Full life-cycle models are large and complex; partition and consistency mechanisms should also be provided.
- The number of relationships among elements of the model is quite large.
- Different levels of granularity of data and process need to be supported.
- The formalism language typically imposes constraints on the models.
- Evolution support is essential since the models evolve.

More recently, we explored implementations in advanced data management platforms. The PMDB+ prototyping investigations were conducted using the VBase object-oriented (O-O) database system [AH87]. The modeling lessons came as a result of the experimentation with the extended E-R formalism to describe the PMDB+ model; the implementation lessons came as a result of the O-O prototyping and the construction of a PMDB+ Viewer tool which supported execution and viewing of the model.

Those lessons learned include:

- *Designing for generic purposes.* Our experience has shown that generic models and generic environments are necessary if they are to support companies like TRW where projects vary from one another. Thus, the PMDB model had a generic design assumption in order to be applicable across projects, and to minimize changes. Examples of entities are: *person, milestones, software component, tasks, documents.* It did not support specific techniques, methods or elements. Those were to be instantiated or refined for project specific purposes. Thus, the model formed a kernel or a base model for the life-cycle. This base model was applied in different contexts and its generic features held well. It was successfully extended in support of specific techniques such as CoCoMo [Boe81]; new types were easily added and mapped via the relationships to existing types. It is worth noting that most of the attributes needed by the CoCoMo technique were already in the model or were easily added, e.g., number of lines of code, analyst and programmer capabilities, software required reliability, and the values of those attributes were easily accessed via the relationships.

- *Modeling of behavior and flow of control.* We modeled process behavior mostly by means of operations associated with the various types. In the current model, the ordering of execution of operations is not prescribed, even though some operations enforce constraints (which implicitly may prescribe ordering). The specification of process flow of control should be provided in process models and definition mechanisms but not prescribed in generic models, because they will vary from project to project and as the model evolves. The specific ordering can be defined when instantiating generic processes into instances of process-based environments.

- *Separation of model from implementation.* We have found necessary to distinguish between a process model and its implementation. Examples of such need include: to be implementation independent, to serve as a user interaction paradigm closer to user's perception, and to support process tailoring by non-expert users. We contend that semi-automated means for translating between process models and their implementations are necessary in order to support tailorability, extensibility, and time-constraint needs. This separation between model and implementation has proven very desirable. It allowed us to distinguish between the formalization of the life-cycle process and the details of the implementation platform. The semi-automated mapping also allowed for the extensibility of the model with minor changes to client applications which use the process implementations.

  The PMDB+ model, being itself an object-based system, mapped easily to the Vbase object-oriented model with the state and behavior of each PMDB object type encapsulated in a separate Vbase type. In our implementation, however, additional operations were associated with the PMDB types in the object base; they were used for implementation purposes and not included in the model. In order to semi-automate the mapping, a notation was defined to distinguish PMDB+ operations from other support operations associated with the types; this way only the PMDB types were retrieved for user consumption. For this and other reasons, the ability to define sub-schemas to constrain access to (possibly intersecting) subsets of operations is necessary in underlying platforms.

- *Mapping relationships.* The full model is large and complex requiring sophisticated semantic constructs. The model also exposed the large number of relationships which are needed and frequently used for navigational purposes. That indicates that relationships should be treated as first class citizens in any system supporting the implementation of such models. A known deficiency of the object-oriented approach is the fact that relationships are not first class citizens. Since relationships are not primitive constructs in object-oriented models, we emulated them using properties; that required that we selected among different strategies. Choosing those strategies implies trade-offs with respect to: support for properties such as *referential integrity*; the facility of using navigation/retrieval capabilities; and the complexity of the generic mechanisms for mapping the model into the implementation.

- *Types as meta-types.* The meta-type notion of the object-oriented approach, where all aspects of objects including their types are represented by typed values in the system (with associated properties and operations), was key to allowing us to write generic mechanisms for type and value retrieval. These generic mechanisms are an important step towards supporting type evolution with minimal impact on the environment.

- *Object management external interfaces.* Program callable interfaces are crucial in support of multi-lingual environments and in support of environment evolution. One of the serious short-comings of the selected system was the lack of a program callable interface to the object store. To gain access to the DBMS facilities, one had to write programs in its own language. This restriction made access to object base from other languages besides C extremely inconvenient. Integration of persistence with a process programming language is conceptually appealing, but issues pertaining to multi-lingual support, full transparent support for persistence, and transaction management are still yet to be resolved in an integrated manner.

- *Architecting for evolution.* For evolution purposes, it is essential to minimize the dependency of process code on underlying platforms. Towards this goal, we used the PMDB+ model as a conceptual interface technique, and defined and implemented a PMDB+ model generic interface. This interface provided a C interface to the PMDB+ model stored in the object base; its clients do not

need to know about the specific system or language used to store and access the model. This allows a possible change to a new storage system or the incorporation of multi-processors without changing the clients. Two key object-oriented features for the development of the generic mechanisms were the equivalence of data and meta-data and dynamic binding. The generic interface also supported type evolution with minimal change to the application code.

- *Multi-lingual component communication.* Incompatibility between multi-lingual environment components is a critical issue of software development environments that needs to be addressed. During the construction of the PMDB+ Viewer, an exercise to interface an Ada UIMS component with the the object base's language did not succeed since their run time environments could not co-exist within the context of one Unix executing process. This conclusion led to further exploration of a client-server architecture to provide for multi-lingual/multi-type communication. These explorations fostered a collaborative effort between TRW and University of Colorado towards the development of a communication model to achieve a language-independent interprocess communication mechanism; this model later led to the development of the Arcadia Q system [MHLO92].

- *Separation of specification and implementation.* Separating the external interface of a type from its internal implementation details, similar to Ada and other languages supporting abstract data type concepts is widely considered as a desirable characteristic. This way, client applications do not need to know about their internal representation and method code can be modified without recompilation of the specification for that object or any client which uses it.

- *Run-time binding of methods.* Late-binding or dynamic binding is extremely desirable for extensibility purposes since it enables different objects to respond differently to the same operation. Based on the direct type of the operation invocation's first argument, methods corresponding to that direct type will be dispatched. We made use of this feature extensively in the implementation of generic mechanisms.

- *Trigger invocation and its interaction with transactions.* Mechanisms such as triggers were extremely useful in support of the implementation of conditional event execution. In our case study, triggers can be attached to operations as well as properties. A trigger can be invoked at the initiation or completion of a single operation. When a trigger is attached to a property, it can be invoked when the property is initialized, fetched, or updated. Using triggers with concurrency code led to interesting problems:

  - A trigger may be activated due to a change of a property value. However, if that change was caused within a transaction which was later aborted, the trigger cannot be rolled back.

  - There was no support for deferring the activation of a trigger until the end of a transaction. There should be some control over whether the trigger is to be invoked immediately or deferred until transaction commit time.

Other not so commonly found desirable OM capabilities in support of process implementations are as follows:

- *Semantic representations in type definitions,* i.e., the ability to specify type or object semantics and constraints as part of their type specifications.

- *Dynamic creation of object types, attributes and relationships,* e.g., the ability to support introduction of new PMDB object types and relationships from an executing process program.

- *Access control mechanisms associated with types, operations, and properties,* i.e., the provision of an access control facility for specifying access constraints on, minimally, object instances.

Acknowledgements

The author would like to acknowledge E. D. Stuckle and C. Shu, co-authors of the models; and I. Thomas, for many interesting and valuable discussions about the design and implementation of life-cycle models. This work was supported by the Defense Advanced Research Projects Agency/Information Systems Technology Office, ARPA Order 7314, issued by the Space and Naval Warfare Systems Command under contract N00039-91-C-0151.

References

[AH87] T. Andrews and C. Harris. Combining Language and Database Advances in an Object-Oriented Development Environment. *SIGPLAN Special Issue (also in OOPSLA'87 Conference Proceedings)*, 22:430–441, December 1987.

[Boe81] B. Boehm. Software Engineering Economics. In *Prentice Hall, Inc.*, Englewood Cliffs, NJ, 1981.

[MHLO92] M. Maybee, D. Heimbigner, D. Levine, and L. Osterweil. Q: A Multi-lingual Interprocess Communications System for Software Environment Implementation. Technical Report CU-CS-92, Department of Computer Science, University of Colorado, 1992.

[Pen86] M.H. Penedo. Prototyping a Project Master Database for Software Engineering Environments. In *Proceedings of the 2nd ACM Software Engineering Symposium on Practical SDEs*, Palo Alto, CA, December 1986.

[Pen93] M. H. Penedo. On the Extensibility of Common Models in PSEEs. In *Proceedings of Process Evolution Workshop*, Montreal, Canada, January 1993.

[PS84] M.H. Penedo and E.D. Stuckle. Integrated Project Master Database - IR&D Final Report. Technical Report TRW-84-SS-22, TRW, December 1984.

[PS85] M.H. Penedo and E.D. Stuckle. PMDB - A Project Master Database for Software Engineering Environments. In *Proceedings of the 8th International Conference on Software Engineering*, London, England, August 1985.

[PS91] M.H. Penedo and C. Shu. Acquiring Experiences with the Modeling and Implementation of the Project Life-cycle Process - the PMDB work. *IEE and British Computer Society Software Engineering Journal*, September 1991.

[Tay89] B. Taylor. The SLCSE Environment Database. In *Proceedings of 1989 ACM SIGMOD Workshop on Software CAD Databases*, February 1989.

Workshop Overview

K. Narayanaswamy, J.C. Franchitti, and R. King

August 22, 1994

The workshop on databases and software engineering was held on the two days preceding ICSE '94. There were 32 attendees, from 8 countries. Most of the attendees were from the Software Engineering community. Each attendee was required to submit a paper in order to be admitted to the workshop; a couple extra people were admitted on-site at the last minute, since there was extra space in the room. There were no paper presentations; rather, the workshop was organized into a series of in-depth discussions centered around specific research questions. In each case, the research question was motivated by a brief talk delivered by one of the workshop participants.

The authors of this overview would like to point out that this summary is our impression of what was said at the workshop. We apologize to any attendee who feels that his or her statements have been misrepresented.

1 Uses of Conventional Databases in Environments

The first talk, by Alberto Mendelzon, described work done in the context of the AT & T 5ESS switch system. Conventional databases were used to store information regarding software project management, testing, reuse, and archeology. Mendelzon's talk explored some of the major issues and tensions in using database technology:

- Organizing databases for exploration and navigation rather than queries.
- Performance trade-offs of expressive query languages.
- Encapsulation of information in objects versus use of relations.

The research topics raised by Mendelzon were as follows:

1. Incremental query computation and display: small changes to data (e.g., change to a single function in a large system) should not force expensive queries to be computed from scratch.
2. Visualization and animation techniques need to be developed for temporal queries. This could involve development of a set of formalisms for visualization and abstraction.
3. Databases need to be integrated better with the software environments and software processes.

In the ensuing discussion Mendelzon focused on the last two issues. Additional concerns were raised, some of which are known shortcomings of databases in accommodating the software engineering domain. These include the large number of types of objects in the software engineering domain, versioning support, and flexible transaction support.

One of the issues that received attention in the discussion was the following: *Exactly what is stored in the database?* Presumably software objects of very different granularity (variables, types, statements, functions, procedures, modules) will be in the database. All of these have relationships to other objects. Points were raised that neither traditional relational databases nor object-oriented databases were adequate to handle these kinds of objects. In addition, some of the information regarding software objects is likely to be unstructured. Techniques for information retrieval in other domains (e.g., markup languages, embedding schema with all data, etc.) seem to be relevant to querying such data.

In looking more broadly at the question of how the software engineering domain was different from other database application domains, it was felt that perhaps if software engineers understood their processes as well as bankers and airline reservation managers, then databases could be better tailored to meet those requirements. However, in the interim, there also seemed to be a general consensus that database management systems are simply not packaged to be as customizable and "open" with pluggable components implementing orthogonal features that a person can select from. Such an architecture would afford the kind of flexibility needed to deal with the software engineering domain.

2 Adapting Databases for Software Engineering: The GoodStep Project

Wolfgang Emmerich described the GoodStep Project, an effort to use the O2 object-oriented DBMS in the software engineering domain. Software objects are stored as abstract syntax graphs. New tools can be built in the O2 language itself, and existing tools are handled by building O2 wrappers for those tools, so that they can be integrated into the environment.

This presentation, more than any other at the workshop, seemed to generate significant discussion; for most of us, it was the first true partnership that we were aware of, where the goal was to get database and software engineering researchers collaborating actively on a substantial development.

What is interesting about the GoodStep experience is that certain key capabilities had to be added to the O2 DBMS before it could be used in this domain. This work could provide more general guidelines about how to architect software engineering databases in the future. The key capabilities that were added to O2 were:

- Versioning of objects.
- Active database capabilities.

The discussions following Emmerich's talk centered around questions of boundaries between the various components within software engineering environments. For example, should knowledge of state change be built into tools or into the logically centralized DBMS? Good arguments can be made in favor of each choice. The present problem is that databases

fold all such capabilities into a single, monolithic bundle. The workshop consensus seemed to be that all basic, primitive capabilities must be part of the database, but the databases should be componentized with standard interfaces, so that different kinds of mechanisms and policies can be realized without undue difficulty[1]. Unfortunately, database systems are simply not architected as a set of cooperating but replaceable components, with well-defined interfaces. However, it was noted that many database researchers and vendors are now moving toward such Object Service Architectures (OSAs), and it is expected that some of the so-called "next-generation" object and object/relational DBMS's should be much more flexible in this regard.

3 Persistent and Database Programming Languages

In sharp contrast to other presenters, Ron Morrison and his colleagues (who paraded up one after the other with surprising discipline with respect to using up time) presented arguments in favor of using persistent programming languages as the vehicle for creating tailored database applications for software engineering that could be superior to using standard databases in several ways:

- Applications in persistent languages will always perform better, because they can be optimized.
- Persistent language applications can be engineered to evolve more gracefully using ideas such as change absorbers, automatic and partial transmitters, etc.

Morrison and his colleagues argued that starting from any traditional DBMS, whether relational or object-oriented, would be the wrong choice because all kinds of inflexibility is inherent in the database, and one must live with the design decisions that are hard-wired into the DBMS. It is better, the argument went, to start with a uniform persistent foundation provided by a persistent programming language.

The discussion following the "Morrison, Inc." talks centered on the fact that, with persistent programming languages, one simply does not have any well-defined, reusable building blocks or components in the environment. Hence, there is likely to be a lot of wheel reinvention – because there is no current solution to program reusability. For example, DBMSs already have machinery to support a team of users through transaction mechanisms, whereas with persistent languages everything must be programmed from scratch. Many in the group opined that the community needs to better understand the trade-offs involved so that one can sometimes use off-the-shelf components and build some components with persistent programming languages.

In a later talk, K. Narayanaswamy presented notions that originated in DBMSs that could usefully be incorporated into "regular" programming languages. These features included schemas (domain models), atomicity of collections of state changes via transactions, consistency through use of integrity constraints, and event driven computations. It was

[1]We note that at this point in the workshop, questions about monolithic DBMS architecture had already emerged as a key discussion point, as a result of Mendelzon's talk.

posited by the speaker, based on his group's experiences with relational abstraction extensions to programming languages, that these kinds of features were very useful in general purpose programming languages.

Vigorous discussions ensued. Some argued that traditional databases simply did not afford the sophisticated typing mechanisms afforded by programming languages. As a result, it is hard to build general purpose applications on top of database management systems. Others argued that databases were inherently multi-user, and notions of transaction and data integrity made no sense in any other case. Yet, with the advent of persistent and database programming languages, such rigid distinctions are becoming increasingly hard to justify.

4 Impact of Process on Database Support

Israel Ben-Shaul described work on the Marvel process-centered environment at Columbia University, analyzing the database and its role in such environments. These projects have created their own object stores, including active database capability and flexible transaction management to support teams. As with the GoodStep Project, it is interesting to note that no off-the-shelf database or system satisfied the requirements of these projects.

The discussion following this presentation examined how the introduction of process might impact upon the requirements for a software engineering database. Some in the audience vigorously questioned whether introduction of process materially changed anything from the perspective of database support. However, it was clear that support for process implies events as first class objects, which usually has an impact on the database support, because the underlying database must be willing to notice events and notify the environment about events. Ideally, in any environment, one would like the ability to replace one database with another. Unfortunately, this usually alters basic process-related capabilities – for example, mechanisms for state changes and events. Once again, this seems to argue in favor of database components with well-specified state-change and event protocols rather than a monolithic database, which is inscrutable and inflexible with respect to these capabilities.

5 Interoperability Concerns in Software Engineering Databases

Nabil Kamel described one kind of interoperation involving the use of information retrieval techniques to formulate queries spanning across multiple, heterogeneous repositories. Some of the techniques he described (based on markup languages) were noted to be very good at extracting useful semantic information from unstructured data. With Kamel's work, users could not directly update the data in the repositories – whereas handling distributed change is clearly a major concern of software engineering environments. Nevertheless, some of the issues raised in this were quite relevant to software engineering.

The state-of-the-art in heterogeneous database work is very much *Download and Read.* However, one can envisage adding more sophisticated support for distributed database update, incorporating support for sophisticated transaction management required by software

engineers. Other research issues were also raised during the discussion, such as how one might add support for updates to Kamel's scheme, propagation of database updates to remote databases, schema integration issues in a network of databases, and whether it is reasonable to require a centralized (meta)model which has information about the data in each of the individual repositories, etc.

Lee Osterweil and Peri Tarr described the Arcadia Software Environment Project, and its experiences vis-a-vis interoperation of heterogeneous components. There is heterogeneity (and need for interoperation) at the level of storage management, at the level of database services, and at the level of language interfaces (based on abstract data types) to object-management services. A lot of experimentation has been carried out on interoperation within this project including the use of centralized databases, federated databases, multiple stand-alone databases using RPC-based communication, and ad-hoc tool wrappers.

This presentation, like the GoodStep project, generated much active discussion, which focused on what might be the general lessons of the Arcadia Project. For example, are there general guidelines about the level at which interoperation should occur and the characteristics of each level? What are the trade-offs of the different mechanisms for interoperation that ARCADIA has experimented with?

Another interesting point was that, in contrast to GoodStep, the Arcadia Project decided not to use tool wrappers for interoperation because, in general, wrappers can be difficult to build for certain tools (e.g., interactive editors).

During the discussions, it was brought to light that the Arcadia Project promotes interoperation at many levels of the architecture, by description of interfaces as abstract data types. In general, Arcadia, because of its emphasis on process, strove for interoperation at the language level. Perhaps because of this, interoperation at the artifact level, which seems desirable, is only now becoming a serious focus of attention.

6 Events and Transactions

The workshop's last talk was provided by Andy Schurr. The talk focused on events and transactions. In particular, he examined the issue of what would happen when a transaction is aborted. The scenarios he discussed were as follows:

1. **Scenario 1:** triggered action modify the same database:
   - Aborting transaction undoes side effects of triggered actions.
   - Database management system aborts transaction without raising events (i.e., no event is needed).
2. **Scenario 2:** triggered action modify foreign database.
   - We have to abort transaction of foreign database.
   - Database management system modifies "foreign clients" with "start/commit/abort" events (1 event is needed).
3. **Scenario 3:** triggered action modify all kinds of data.

- We have to undo effects on foreign data step by step.
- Database management system has to generate sequence of inverse events to compensate for actions that have to be undone (many events are needed).

After this talk, the discussions centered around how events could be used to keep a federation of databases consistent – for example, each event can modify a foreign database by exporting partial results. Of course, in this case, both the database management system and applications programs must tolerate inconsistency for the duration when the events are performing their updates on databases.

7 Retrospective and Assessment

Database management systems and database-like notions have been seen as relevant to software engineering for some time. Some of these ideas first crystallized at the "NAPA" workshop (held in Napa Valley in 1989, with approximately an even representation by software engineering and database researchers). For example, the NAPA workshop laid out the requirements of a software engineering database at a fairly high level of abstraction – a laundry list of desired features that traditional databases, at that time, were incapable of providing. We attempted to avoid discussing these broad requirements in Sorrento, in order to focus on techniques to address these requirements.

At this workshop, tellingly, there was little dispute on the **requirements** of a software engineering database or bickering about terminology and competing world views. There are two explanations for this phenomenon:

- The composition of the group was much more homogeneous, with most in the group being software engineers. Very few in the group considered themselves "mainstream" database researchers. This greatly reduced the opportunities for clashes.
- At least among the software engineers, there is some consensus on the basic requirements of a software engineering database. The emphasis, this time, was clearly on how one might go about building one of these things.

At the Sorrento workshop, it was evident that considerable amount of work had indeed been devoted to development of databases to support software engineering. There are several major projects such as Arcadia, GoodStep, Marvel, etc., which have already examined issues of object management in depth. The Marvel Project has chosen to build its own object management system, because no general purpose system was found to be flexible enough for a process centered environment. However, in the GoodStep project, there is a large effort to extend a general-purpose database, O2, with capabilities (such as versioning, and event-based computation) to make it suitable for software environments. Arcadia has worked with existing object bases and has built its own, with more of a focus on heterogeneity and interoperation, the assumption being that there will never be "one" software environment database.

From a research stand-point, the following themes emerged from this

- If general purpose databases are to serve software engineers, they must be architected not as a single monolithic, "take it or leave it" bundle of features. Rather, databases must be architected as a potentially reconfigurable set of components, each supporting some ideally orthogonal database feature, with standard interfaces. This kind of design has the potential of allowing software engineers to customize databases to their tastes by replacing some components with others as their needs dictate. The results of the GoodStep Projects and similar efforts to adapt general purpose databases to software engineering will be interesting to monitor.

- The other thrust of research (embodied in the work on persistent programming languages and database programming languages) essentially abandons the hope that "standard" database technology will ever completely support software engineers, choosing instead to incorporate useful database-like notions (e.g., persistence, relations, queries, etc.) into programming languages. The problem with this approach is that it does not seem to leverage off of pre-existing components, advocating that all repository support be programmed from scratch.

- People building process-centered software engineering environments are inducing their own database requirements. This workshop examined the impact of process on database support at length. An exemple of this kind of situation is the MARVEL Project, whose developers created their own objectbases and repositories because they could not find adequate support for process notions in standard database technology.

- Heterogeneity and interoperability of various kinds of repositories and tools has already emerged as a major problem in large environments. This has spawned several research issues including the need for interfaces and standardization of environment architectures.

Going in, the goals of the Sorrento workshop were broadly to follow-up on the NAPA workshop, assess the state of the art, and, in the best case, develop a joint research agenda for the software engineering and database communities.

In terms of the above goals, the Sorrento workshop was able to conduct a reasonably detailed assessment of the state of the art in building databases for software engineering. However, no explicit joint research agenda was agreed upon by the workshop, though some issues, such as the monolithic architecture of DBMSs and interoperability of heterogeneous components, imply directions for future work. The lack of a precise agenda was largely a result of the rather minimal representation by the database community. And, because the workshop participants were mostly software engineers, the ultimate impact of the workshop on the larger database research community is unclear at this point.

INFORMATION PROCESSING 89, G.X. Ritter (ed.)
Elsevier Science Publishers B.V. (North-Holland)

Report on the 1989 SOFTWARE CAD DATABASES WORKSHOP

Lawrence A. ROWE

Computer Science Division-EECS, University of California at Berkeley
Berkeley, CA 94720, U.S.A.

A workshop was held to develop a better understanding of the features and database requirements of software development environments. It was organized into a series of moderated discussions between all participants.

The major conclusion was that software development tools need most features found in commercial relational database systems and many features found in next generation object-oriented database systems currently being developed. Specific features required include: object-oriented data models, navigational and set-oriented query languages, complex object support, long transaction support, derived data support, and alerters. It was also apparent that better logical and physical database design tools would significantly improve the development of these new systems.

1. Introduction

A two day workshop on the topic of software CAD databases was held in Napa California on February 27-28, 1989. Approximately 10 people from the database community and 40 people from the software engineering community attended the workshop. The group included a mixture of people from academia and industry. Attendance was limited to encourage dialog between the two communities. The attendees were selected by a program committee that read position papers submitted by people who wanted to participate. These position papers were published in a workshop proceedings [1].†

The goal of the workshop was to develop better understanding in the software engineering and database communities about the database requirements for software CAD databases,‡ the capabilities of existing commercial database systems (DBMS), and the capabilities of next generation object-oriented database systems (OODBMS) that are currently being developed. The workshop was organized into four sessions that covered the following topics (the session leader is listed in parentheses):

SDE Services
(B. Boehm, TRW)
Database Requirements for SDE's
(W. Paseman, Atherton Technology)
Alternative DBMS Architectures
(D. DeWitt, U. of Wisconsin)
Workshop Summary
(L. Rowe, U.C. Berkeley)

Each session began with a short presentation on the issues and followed by a moderated discussion. A designated person took notes during each session. These notes will be published at a later date.

This paper summarizes the session discussions and the conclusions the group drew at the conclusion of the workshop. It was not possible to have all attendees read and comment on the paper due to tight publication deadlines so I apologize in advance for any errors or omissions. The remainder of the paper summarizes the discussions in each session.

† A limited number of copies of the proceedings can be ordered from Sharon Wensel who can be contacted by phone (415-642-4662), email (wensel@postgres.Berkeley.EDU), or by postal mail at the same address as the author.

‡ One problem that immediately became apparent is that there is no generally agreed upon term for programming environment tools. The term software CAD (SCAD, pronounced "ess-cad") was suggested by Bill Scherlis at DARPA. In the software engineering community people use other terms including: integrated project support environments (IPSE), software engineering environments (SEE), and software development environments (SDE). In the remainder of the paper I will use the term SDE.

2. SDE Services

This session addressed the services that a SDE system should provide. The goal was to identify the data that should be stored in a DBMS and the kinds of operations that might be performed on that data. Following this discussion, several people presented short "war stories" about their attempts to build a SDE's on a DBMS.

A SDE database must include all information relating to the software lifecycle process. This information includes:

1. Product data (e.g., specifications, code, documentation, etc.).
2. Resource data (e.g., people, facilities, equipment, budgets, etc.).
3. Management data (e.g., schedules, action items, problem reports, etc.).

Figure 1 shows several queries that might be answered by querying this database. The first query involves complex queries over data that is derived from the data stored in the database. The second query may require a change to the product definition (i.e., application schema change). The third query triggers an automated activity. The fourth query shows an example of a fine granularity query on the source code. And finally, the fifth query is an example of a fuzzy query.

Query 1
List the programmers and managers of all tasks on the critical path with over 5 days of slippage in their current milestones.

Query 2
Take the "computer experience" cost driver attribute for each module in the system and split it into the "computer experience" for the host-system and target-system.

Query 3
Perform an appropriate set of regression tests and report the possible adverse side-effects of every module change.

Query 4
List all exceptions that could be raised by the system for which there is no exception handler.

Query 5
If we change the security level of a specific piece of data, describe how it will effect the security of the complete database.

Figure 1: Example queries.

The database people at the workshop claimed that queries one, two, and four can be solved with conventional DBMS's assuming that reasonable database designs are used. Queries three and five, on the other hand, are much harder. The ensuing discussion identified several issues related to database support for SDE's including the fact that current commercial DBMS's provide inadequate support for dynamic changes to the database design (i.e., schema evolution), derived data (i.e., data computed from data stored in the database), complex objects, and version control.

Several people presented "war stories" about their attempts to build SDE's on a DBMS. William Paseman described the evolution of the Atherton Technology products from a programming language environment tool to an integrated project support environment. The programming language tool supported multiple user access to source code and cross-reference data. The IPSE added support for management control data. Atherton has built an object storage system that supports version and configuration management. They concluded that a programming language environment tool does not require sophisticated database services (e.g., sharing, access control, and associative queries) but that it did need good data modelling, efficient support for fine granularity objects (i.e., abstract syntax tree nodes) and navigational queries (i.e., get next object given an object identifier (OBJID)[§]

Dennis Heimbigner from the University of Colorado at Boulder described his experiences developing a system that manages requirement specifications (REBUS) on top of the Cactis research prototype DBMS [3]. The novel feature of Cactis is that it supports automatic recomputation of derived data in the database.[+] Heimbigner had to develop an interface between ADA and Cactis. He described a variety of problems with interfacing an existing programming language to a DBMS that are well known in the database community (e.g., type compatibility, incompatible data models, etc.). Other problems he described related to the fact that Cactis was a research prototype that did not provide all the functions a commercial DBMS provides (e.g., dynamic schema changes, secondary indexes, sophisticated query optimization, and transaction management). This discussion raised an issue that came up several times during the workshop. A SDE has many database requirements that can be satisfied by features found in different DBMS's. The problem is that no single DBMS provides all the required features.

[§] An object identifier is a unique identifier assigned by the DBMS that never changes [2].

[+] Cactis uses an attribute grammar to specify the derived data computation. Other research database systems are exploring the use of rules to specify derived data (e.g., POSTGRES [4] and STARBURST [5]).

Mark Dowson, currently at the Software Productivity Consortium (SPC), described two systems: one built on a custom DBMS and one that is being built on a commercial DBMS. The first system, called ISTAR, was built on a federated DBMS, that is, a collection of independent communicating DBMS's. The advantage of this approach is that it may be possible to integrate existing tools into a SDE by interfacing the tool-specific DBMS to the federated DBMS. The disadvantage is that a federated DBMS is really a distributed heterogenous DBMS. Consequently, the standard distributed DBMS problems must be solved (e.g., distributed query optimization, distributed transactions, replicated data, and catalog design and maintenance) [6]. In most cases an independent DBMS cannot be changed so it may be impossible to implement all required facilities (e.g., distributed transactions require that the federated database master process be able to access the local database lock tables or to set timeouts on transactions to implement distributed deadlock detection). In addition, Dowson noted the problems associated with building a custom DBMS. Specifically a DBMS is a large complex software system that requires considerable resources to build and maintain. He also described an effort at SPC to use a commercial SQL-based DBMS to build a SDE. The primary problem that they have encountered is that the conventional transaction model is not appropriate for SDE's. This topic is discussed in more detail below.

The final "war story" was presented by Ian Thomas from GIE Emeraude. He described the PCTE project's Object Management System (OMS). A major goal of PCTE is to create a tool interface abstraction that allows existing tools to be integrated with the SDE. OMS has an entity-relationship model with some object-oriented capabilities (e.g., attribute and relationship inheritance). Two problems were encountered. First, interfacing existing tools to a SDE is a very hard problem. And second, developing a good database design that supports tool integration is difficult. Several people who have tried to build SDE's on databases commented on the difficulty of developing good database designs. The importance of good design tools and the ability to rapidly change a design are well-known problems in the database community.

While some progress on the database design problem has been made in the past decade, too much expertise and effort are required to build a complex database application. Database systems should monitor access patterns and automatically change the storage structures so that queries can be executed efficiently. In addition, better support is needed to reduce program and data translation required when the logical database design is changed.

Extensible data model.
Support for meta-schemas (i.e., schemas stored as data).
Operations stored with objects and encapsulation.
Explicit relationships.
Support for derived data (i.e., rules).
Transitive closure queries to access hierarchical data.
Multiple programming language interfaces.
Query optimization and indexing.
Complex object support.
Support for large data sets.
Version support.
Automatic selection of storage structures.
Comprehensive access control facilities.
Bulk data load and unload.
Short and long transaction support.
Crash recovery.
Undo facility.
Portable DBMS (i.e., it must run on many platforms).
Client-server architecture.
Distributed database support.
Acceptable performance.

Figure 2: SDE database requirements.

3. Database Requirements for SDE's

The second session explored in more detail some of the database requirements that were identified in the first session. Several lists of database requirements for SDE's have been published. Figure 2 shows a list developed by Maria Penedo from TRW that was discussed during this session. While a consensus did not emerge, several different viewpoints did emerge during this discussion. First, several database people argued that most of these requirements have already been addressed by commercial relational DBMS's or are being addressed in one of the research prototypes that are currently being developed. A second viewpoint was offered by some of the software engineering people who were unsure that a future, unknown, and unproven DBMS that would solve the SDE problem will be forthcoming within a reasonable timeframe. Finally, others argued that a radically different open database architecture was needed that would allow programming languages to selectively use powerful database features (e.g., associative access, crash recovery, etc.) on data in the database and non-persistent data created by the program. This last proposal is discussed in more detail in the next section.

The remainder of this session covered a variety of topics on transactions, query optimization, data models, and historical databases. The most interesting discussion centered around the topic of transactions. Gail Kaiser from Columbia University presented a short overview of the

capabilities of a transaction system and the conventional DBMS strategies that are used to implement these capabilities. Several problems were identified including the following.

1. SDE's need more capabilities than a conventional transaction system provides. Specifically, a SDE must be able to manage inconsistency. For example, a tool might require consistency within a complex object such as a program module but inconsistency between complex objects such as the other modules that use the module being modified by the tool. Another example is that a tool may want to enforce consistency, but delay notification to others that an update has been made to the database.
2. SDE's need to support multiple processes within a single transaction. For example, two tools running on a workstation may be showing different views of the same data (e.g., the source code for a procedure and the call graph for the system). Updates can be made to the data through either tool but the database should see them as one transaction.
3. A SDE needs efficient support of different types of transactions. Some applications read and update relatively little data in a transaction. These transactions are called *short transactions*. Other applications execute transactions that run for a long time while the user browses and updates many different objects in the database. These transactions are called *long transactions*. Conventional DBMS's provide excellent support for short transactions. However, these systems have trouble with long transactions because users are prohibited from accessing the data read and written by the transaction.

Kaiser described several approaches that researchers are experimenting with to solve these problems. The first approach uses nested transactions [7]. A nested transaction allows a transaction to spawn a sub-transaction that can commit before the parent transaction commits. The Sun Network Software Environment uses nested transactions [8]. In both systems a user can make several changes to a virtual copy of the database. These changes can be viewed as nested transactions on the virtual database within the larger transaction that will be completed when these changes are merged back into the main database. This approach solves problems 1 and 2 above.

A second approach to solving some of these problems is to use naming domains to control access to the database. In a naming domain, all versions of objects are retained. A user operates on a "configuration" that defines a set of object versions. A transaction is executed with respect to an initial configuration. An update transaction that commits creates a new configuration. Naming domains can be used to solve problem 1 above, namely, managing inconsistency between complex objects. This approach is being investigated in the COSMOS system [9].

A third approach is called *participant transactions* [10, 11] The idea is that several processes can participate in the transaction. Transactions are named so that a process can join a running transaction. Consequently, multiple processes can execute within a single transaction (i.e., it solves problem 2 above). Each process sees the database with all participant's updates, but the rest of the users do not see them.

A fourth approach is to use commit-serializability (CS) transactions. CS allows a transaction to split into several distinct transactions as long as they have disjoint write sets (i.e., the set of objects the transaction has updated) and the read set of each new transaction is disjoint from the other new transactions being created in the split. These new transactions can commit or abort independently or they may join with any other transaction in the system to create another new transaction. All transactions that commit are serializable, but they may be completely different than the set of transactions that were initially created [12]. The idea is that transactions are created, split, merged, and committed as the user examines and updates the database. CS transactions can be used to solve problems with long transactions.

Lastly, database researchers are exploring another approach to solving the long transaction problem, called *sagas*. A saga is a long transaction that can be broken up into a collection of sub-transactions that can run at the same time with other transactions. These sub-transactions are related to each other and all must commit for the saga to commit. Sub-transactions are non-atomic which means that database updates made by the sub-transaction can be undone at a later time by a "compensating transaction" that must be defined for each sub-transaction. The advantage of sagas is that more concurrent access is possible because sub-transactions can be completed and the resources they control can be released [13].

Most people agreed that there is still much work to be done in this area.

Another topic discussed in this session was the requirement that a rule in the database invoke some action when the predicate becomes true. For example, a manager might want to be notified when the bug count in a particular part of the system had reached a certain threshold. This capability is called an alerter in the database community [14]. Few, if any, commercial DBMS's support alerters.

4. Alternative DBMS Architectures

The majority of this session was used to allow the developers of various database systems to describe their systems. The following systems were discussed:

Software BackPlane† (Atherton Technology) [15]
Cactis (University of Colorado at Boulder) [3]
EXODUS (University of Wisconsin) [16]
Gemstone (Servio-Logic) [17]
Iris (HP Laboratories)
Observer/Encore (Brown University) [18]
POSTGRES (University of California at Berkeley) [4]
A Yet to be Named Product (Ontologic) [19]

Several themes emerged from these presentations. First, all of the systems are object-oriented in the following senses: 1) they provide richer type systems than a conventional relational DBMS, 2) they support some form of object identity, and 3) they support inheritance. Some, but not all, systems extend a set-oriented query language (e.g., SQL) with user-defined procedures and methods and some store methods and procedures in the database.

The second theme was the importance of support for complex objects. Typically, this support includes some mechanism to load an object composed of many objects with different types that are highly interdependent (i.e., they contain many attributes with references to other objects in the complex object) very quickly. Object references are represented by OBJID's that are assigned by the DBMS and never changed. The load process usually translates the database representation of values to an appropriate representation for the program. This translation is called *swizzling*. Most systems convert OBJID's to main memory pointers, called *pointer swizzling*, so that subsequent references can be implemented very efficiently. Main memory performance is critical for many of the applications that these systems are addressing, including SDE's.

A third theme that emerged was that any next generation DBMS must provide all functionality that is provided by current commercial relational DBMS's. This fact was apparent both from the requirements list presented in figure 2 and the discussion during the workshop. Specifically, the DBMS must support associative queries, multiple programming language interfaces, database procedures (i.e., the ability to dynamically link application code into the DBMS process), and conventional transactions.

The discussion then turned to an object-oriented programming system with an integrated database that allows a programmer to use database functionality on any object. The basic idea is that some database functions (e.g., associative queries and atomic operations) should be available on objects created by a program that are not persistent. In addition, these functions should be applied uniformly to across all objects (i.e., persistent and non-persistent). Examples are queries that search for data in the database, in a program cache that holds objects that have been fetched from the database, and non-persistent objects in the program. Another example is that it should be possible to define a rule on database and program objects.

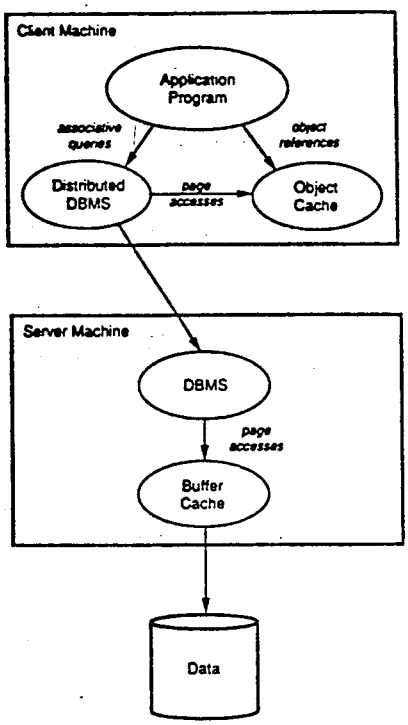


Figure 3: Integrated programming environment architecture.

The software architecture that runs on the distributed system shown in figure 3 was proposed by several people. The object cache holds database and program objects. Object references in the program access this cache directly. Associative queries are handled by the distributed DBMS code in the client machine. This code treats the object cache as another local data manager similar to the DBMS that runs on the server machine. While this architecture is conceptually clean, many hard problems remain to be solved. For example, how does the system optimize a complex query that joins database and program objects where some of the database objects have been fetched into the object cache and modified by the program. Several groups in the database and programming language communities are working on similar systems [18, 20-22].

† Trademark of Atherton Technology.

At the end of the session two issues related to standards were raised. First, someone said that they wanted a better object-oriented data model than the model provided by C++. This issue was raised because most attendees recognized that C++ will be the most widely-used object-oriented programming language model due to the popularity of C. The problem with a C++ data model is the absence of a standard set abstraction, a rules system, and a set-oriented query language. Tim Andrews from Ontologic identified the real problem when he noted that his company had developed a better data model in their VBASE product but that the marketplace was not interested in it.

The second issue raised was whether SQL was the right query language. As with C++, SQL is clearly the dominant query language and it is likely to remain so for a very long time. The problem with SQL is the difficulty of extending it to support new features (e.g., transitive closure queries and complex object support).

5. Workshop Conclusions

The final session was used to produce a list of conclusions with which the majority of attendees could agree. The following conclusions were agreed upon.

1. Both within the database and software engineering communities there are many inconsistent and confusing terms. Everyone who attended agreed that the meeting had been productive in that it exposed some of this confusion and in some cases led to agreement on common terminology (e.g., participant transactions).
2. The development of a SDE, viewed as a database application, requires more programmer control than the business applications that currently make up the majority of applications for a conventional database system. Specifically, there is an urgent need for more functionality (e.g., complex object support, versions, database rules, alerters, transitive closure queries, non-traditional transaction models, better integration of database services and program environments, and schema evolution support) while at the same time providing acceptable performance.
3. Next generation DBMS's will be object-oriented and they will have to provide a superset of the capabilities found in current commercial relational DBMS's.
4. Version management is not well understood and there is no evidence that database systems will provide the required support for the sophisticated version systems required by a SDE.
5. Database systems must provide better support for schema evolution. At one point during the workshop people discussed the idea of the SDE being able to run consistently across major changes to the SDE schema and still answer the kinds of complex queries described above. This capability presents a major challenge to database researchers that might not be achievable.
6. The majority of attendees were skeptical that an acceptable, commercially supported DBMS with all the features required by a SDE will be forthcoming in a reasonable timeframe.
7. Interfacing existing tools to a SDE is a very hard problem and nobody has any good ideas about how to solve it. Some people thought this will be a critical requirement for future SDE's.
8. Lastly, many people agreed that there must be life after C++ and SQL, but everyone reluctantly agreed that the marketplace would continue to make them the dominant languages.

During this session, the group also produced a list of topics that were not discussed during the workshop and came to a conclusion as to whether this exclusion was good or bad. This list included the following.

1. The concept and semantics of *object* was not raised at any time during the workshop. Everyone unanimously agreed that this exclusion was good.
2. There was no discussion of security. Several people noted that this topic is extremely important and that it will have to be addressed eventually.
3. User-interfaces were not discussed. This exclusion was a conscious decision by the group at the beginning that was agreed upon so that we would not be distracted from the topic of databases. There was general agreement that this decision was good.
4. Finally, there was no specific discussion of whether a SDE database must be an integrated database (i.e., that all data must be stored in a single database that might be distributed) or a federated database (i.e., data is stored in different databases that may not support a consistent data model). This topic has been important in the business application community where it has been discovered that a single integrated DBMS that supports all applications and hardware platforms is not available. It remains to be seen whether the same is true for SDE's.

Acknowledgements

I want to thank Dave Dewitt and Lee Osterweil who supported this workshop in many ways. Without their help, it would never have happened. I also want to thank the session note takers: Gail Kaiser, Dennis Heimbigner, and Maria Penedo. I could not have written this report without their input. All errors are my fault not theirs.

References

1. L. A. Rowe and S. Wensel, editors, Proc. 1989 ACM SIGMOD/SIGSOFT Workshop on Software CAD Databases, Feb. 1989.
2. S. N. Khoshafian and G. P. Copeland, "Object Identity", *Proc. 1986 OOPSLA Conf.*, Portland, OR, Sep. 1986, 406-416.
3. S. E. Hudson and R. King, "Object-Oriented Database Support for Software Environments", *Proc. 1987 ACM-SIGMOD Conf. on Management of Data*, San Francisco, CA, May 1987.
4. M. R. Stonebraker and L. A. Rowe, "The Design of POSTGRES", *Proc. 1986 ACM-SIGMOD Conf. on Management of Data*, Washington, DC, June 1986.
5. B. Lindsay, J. McPherson and H. Pirahesh, "A Data Management Extension Architecture", *Proc. 1987 ACM-SIGMOD Conf. on Management of Data*, San Francisco, CA, May 1987.
6. S. Ceri and G. Pelagatti, *Distributed Databases – Principles and Systems*, McGraw-Hill, New York, 1984.
7. J. E. B. Moss, "Nested Transactions and Reliable Distributed Computing", *Proc. 2nd Symp. on Reliability in Dist. Soft. and Database Sys.*, Pittsburgh, PA, July 1982. Available from IEEE Computer Society Press.
8. E. Adams and et. al., "Object Management in a CASE Environment", *Proc. 11th Int. Conf. on Software Engineering*, Pittsburgh, PA, May 1989.
9. J. Walpole and et. al., "A Unifying Model for Consistent Distributed Software Development Environments", *Software Eng. Notes 13*, 5 (Nov. 1988).
10. G. E. Kaiser, "Extended Transaction Models for Software Development Environments", Technical Report CUCS-404-88, Columbia Univ. Dept. of Comp. Sci., 1988.
11. M. Dowson, *Proc. 1989 ACM SIGMOD/SIGSOFT Workshop on Software CAD Databases*, Computer Science Division - EECS, U.C. Berkeley, Feb. 1989.
12. C. Pu, G. E. Kaiser and N. Hutchinson, "Spit-Transactions for Open-Ended Activities", *Proc. 14th Int. Conf. on Very Large Data Bases*, Los Angeles, CA, Aug. 1988, 26-37.
13. H. Garcia-Molina and K. Salem, "Sagas", *Proc. 1987 ACM-SIGMOD Conf. on Management of Data*, San Francisco, CA, May 1987.
14. O. P. Buneman and E. K. Clemons, "Efficiently Monitoring Relational Databases", *ACM Trans. Database Systems*, Sep. 1979, 368-382.
15. W. Paseman, *Proc. 1989 ACM SIGMOD/SIGSOFT Workshop on Software CAD Databases*, Computer Science Division - EECS, U.C. Berkeley, Feb. 1989.
16. M. Carey and et. al., "The EXODUS Extensible DBMS Project: An Overview", Computer Sciences Technical Report #808, Univ. of Wisconsin, Nov. 1988.
17. G. Copeland and D. Maier, "Making Smalltalk a Database System", *Proc. 1984 ACM-SIGMOD Conf. on Management of Data*, June 1984.
18. A. H. Skarra, S. B. Zdonik and S. P. Reiss, "An Object Server for an Object-Oriented Database System", *Proc. Int. Wkshp on Object-Oriented Database Systems*, Asilomar, CA, Sep. 1986.
19. T. Andrews, *Proc. 1989 ACM SIGMOD/SIGSOFT Workshop on Software CAD Databases*, Computer Science Division - EECS, U.C. Berkeley, Feb. 1989.
20. R. Balzer and et. al., "Specification-Based Computing Environments", *Proc. 8th VLDB Conf.*, Sep. 1982, 273-279.
21. W. Kim and et. al., "Integrating an Object-Oriented Programming System with a Database System", *Proc. 1988 OOPSLA Conf.*, San Diego, CA, Sep. 1988, 142-152.
22. L. A. Rowe, "A Shared Object Hierarchy", *Proc. Int. Wkshp on Object-Oriented Database Systems*, Asilomar, CA, Sep. 1986.

Life-cycle (Sub) Process Scenario

for 9th International Software Process Workshop (ISPW9)*

March 1994

Maria H. Penedo
TRW
One Space Park
Redondo Beach, CA 90278

Abstract *This paper contains the ISPW9 scenario or process example. It is a continuation of the International Software Process Workshops' tradition of providing a process scenario as an example of life-cycle sub-processes for specifications in different formalisms and implementation and demonstration in different systems. It was defined to support a demonstration day at the International Software Process Workshop (ISPW9).*

1 Background

This scenario is a continuation of the International Software Process Workshops' tradition of providing a process scenario as an example of life-cycle sub-processes for specifications in different formalisms and implementation and demonstration in different systems.

These examples or scenarios, together with specific guidelines, were designed and planned with many objectives in mind, including:

1. to provide canonical examples as vehicles for distinguishing the distinct process definition approaches.

2. to provide samples of "real life" issues in order to facilitate the explanation of demonstrations and to make those demonstrations applicable to prospective users.

3. to bring about specific technical issues to be addressed by different process formalisms and systems and as a means to experiment with those issues in the various approaches and systems.

The original example, from ISPW6, appears in the Proceedings of the 1st International Conference on the Software Process, held in California, in October 1991 [IEEE Computer Society Press]. Extensions to this example have been proposed in ISPW7 and ISPW8.

2 Introduction

This year's scenario is a revision of the ISPW6 scenario, to take away some of its rigidity, to make it more realistic and tailorable, and to add items related to human-computer interaction and computer mediated human cooperation. This scenario is flexible, i.e., it provides a base scenario which includes named but un-specified procedures and policies. Thus, demonstrators can extend the scenario to demonstrate diverse policies and models, and the strengths of their systems.

This scenario focusses on:

- **Life-cycle aspect:** Problem Reporting and Change Process

*in Proceedings of ISPW9.

- **Theme:** Roles of human in the process and automation support for individual/team activities.
- **Objective:** Demonstrate process execution and how a process-based SEE helps project users in their roles (e.g., project manager, designers, developers, configuration managers) perform their activities and cooperate with each other.

The base scenario appears below, followed by a set of sub-scenarios with recommended themes to be demonstrated together with the scenario. These themes either refine the base scenario by including specific procedures, or list candidate functionalities to be demonstrated.

At ISPW9, there will be a demonstration day preceding the workshop (open only to workshop attendees). Demonstrators are requested to enrich the base scenario with the sub-scenarios. It has been our experience that the demonstrations provide a good source of ideas and concepts to be discussed throughout the workshop.

3 Base Scenario for Demonstration: Problem Reporting and Change Process

- A software project is on-going, with "parts" of the system already designed, codified, tested and baselined (i.e., under configuration management control).
- A problem is reported by a tester on the testing of a piece of the system under development. The project's problem reporting and analysis procedures are then followed and a person is assigned the task of the analysis of the problem. (Note: these procedures can be formal or informal, depending on the type of project. Notification can be effected by mail, by forms, by a tool. The procedures may include rules or guidelines telling who assigns people resources to study which problems and what kind of steps need to be followed.)
- A developer/analyst analyzes the problem and proposes a solution. After the analysis (which can be illustrated via automated process support or assumed to have been done manually), the developer identifies that the problem affects one software module which has been coded, tested and baselined, and possibly also affects some documentation (e.g., design and/or testing documents). (Note: the related documentation can be identified explicitly with help from the system, or implicitly via existing pre-defined rules in the system).
- After some analysis, it is noted that the module to be fixed is currently being (re-)used by two separate users or teams (again how this is accomplished may vary, i.e., the system may flag this issue or this fact may be found explicitly by inspection by a configuration manager or the developer). Those users are notified of the problem and that the module will be changed.
- The change process starts according to pre-established change procedures (which entail assignment of resources, code and/or documentation modification, analysis/testing/review, approval/rejection and new baseline of the module and associated documentation).
- The module is checked out of the baseline according to the CM procedures for change but reuse of the old version continues.
- The module is changed to fix the problem. (Optionally, the fix could be done by two or more separate developers and their cooperation may be illustrated via process support).
- The module is tested (formally or informally). Once the problem is fixed, procedures for acceptance/rejection are followed. Once the module is accepted (i.e., the change does fix the problem and it does not violate any of the requirements), appropriate regression testing on the modules/systems which reuse a prior version of this module can be performed.
- Once all is done, the change process is finalized.

4 Sub-Scenarios.

Demonstrations should explicitly include as many of the following sub scenarios as possible:

1. **Policy Extension.** Specify and demonstrate one or more specific procedures/policies to complement the scenario (preferably performed with automated process support):
   - problem reporting and/or analysis
   - testing procedure/method
   - analysis of a problem using data in system
   - configuration control: retrieval, storage
   - code fix

- problem approval/rejection
- resource allocation

2. **User Role Support.** Demonstrate implicit/explicit support for project user roles (see definition in note), i.e., demonstrate: i) (multiple) user to (multiple) role assignment, either static or dynamic; ii) the impact of actions of one role upon another (i.e., automated cooperation among roles based on process definition); and iii) how roles affect the interaction styles and other aspects of the process.
   Note: *Definition of Life-Cycle User Role (adapted from Webster):* An expected behavior pattern associated with one or more people when executing life-cycle activities (e.g., project manager, configuration manager, developer, system analyst). One person can play multiple roles in a project. For example, when someone is writing code, s/he is playing the role of developer; when s/he is doing configuration management, s/he is playing the role of configuration manager.

3. **Individual Support.** Demonstrate how individuals are guided about what task to do next, how users are made aware of the state of the process, or how the system performs actions as a result of the users' actions. Demonstrations should clearly illustrate how users are aware of the process, how the environment and individuals interact, and what variables control the different modes of interaction.

4. **People Coordination.** Demonstrate coordination of multiple people, including any support for resolution and tolerance of inconsistency.
   In particular, demonstrations can illustrate which aspects of these policies, if any, are hard-wired into their systems, and which can be altered by the particular model, and when the policy selections are made.

5. **Configuration Management.** Demonstrate how software and/or documents are controlled for the purpose of change, and how individuals using a module in their development are made aware of problems and/or changes to that module.

6. **Project/Central vs Individual Coordination.** Demonstrate how the executing process supports both individual and project activities, and how the interactions of those activities are supported/mediated by the system.

7. **Process Changes while in execution.** Dynamically demonstrate changing any of the process definitions supporting the scenario and points 1-5 above, and the effects of those changes.

5 Acknowledgement.

Many thanks to the Program Committee, A. Finkelstein, K. Futatsugi, C. Ghezzi, G. Kaiser, K. Narawanaswamy, D. Perry, for providing ideas and reviewing earlier versions of this scenario. This work was supported by the Advanced Research Projects Agency.

ISPW9 Process Demonstrations - Summary*

Maria H. Penedo
TRW
One Space Park
Redondo Beach, CA 90278

Abstract

A process demonstration day was held at the 9th International Software Process Workshop (ISPW9), Washington, DC, 1994. The objective of the demonstration day was two-fold:

- *to evaluate how different systems and environments support/guide users in the fulfillment of their project activities, and*
- *to bring forth technical issues identified by the different implementors in the context of their formalisms and systems.*

A scenario example was defined [1] to represent issues in the life of real projects and to serve as a common example for demonstration purposes. This document gives some background to the scenario, briefly describes the systems demonstrated, and provides a commentary about the demonstrations. Architecture depictions of those systems appear in the Appendix.

1 Scenario Background

In the last few years, the process community has defined a "process scenario" which describes a sub-set of the software development process activities, to serve as a canonical example for the community. It was done in conjunction with the International Software Process Workshops (ISPW).

In the first years, the scenario was used to understand and compare process modeling notations. In the last two years, this scenario is being demonstrated by existing research and commercial tools and PSEEs; it has served to highlight differences among existing approaches and to identify strengths and weaknesses of such approaches. The first example was defined for the 6th International Software Process Workshop (ISPW) and it appears in [2].

The scenario identified for ISPW9 [1] is a revision of the first scenario to make it more flexible, realistic and tailorable. It also adds items related to human-computer interaction and computer mediated human cooperation, which were the themes of the workshop. It consists of a base scenario dealing with Problem Reporting and Change Process and a set of optional sub-scenarios to highlight, among other things: support for user roles, people coordination, tailorability to specific policies, individual vs project coordination, and process changes.

2 Systems Demonstrated

Eight systems were accepted for demonstration: Hakoniwa, from Osaka University (Japan); LEU, from Lion Gesellschaft fur Systementwicklung mbH (Germany); MVP-S, from University of Kaiserslautern (Germany); Oikos, from Pisa University (Italy); Oz, from Columbia University (USA); Synervision, from Hewlet-Packard (USA); Regatta, from Fujitsu; SPADE, from P. Milano. The latter two systems could not be demonstrated. The systems were demonstrated by H. Iida, V. Gruhn and S. Wolf, C. M. Lott, C. Montangero, I. Ben-Shaul, and John Diamant respectively.

The environments demonstrated are briefly described below. Table 1 shows, for each system, its process formalism, user interaction paradigm, general architectural communication model, and database used. Architecture depictions of those systems appear in the Appendix; they were provided by the demonstrators.

*in Proceedings of ISPW9, Arlie, VA, Oct 94.

Table 1. Summary of Systems Demonstrated				
System	**Process Formalism**	**User Interaction**	**Comm. Model**	**Database**
Hakoniwa	- concurrent, sequential task language - template-based	- graph-based animation - color-based - menu-based activity navigation	multi-client single-server (TCP/IP)	File System (FS)
Leu	- integrated models: Petri-net, ER, Hierarchical	- agenda-oriented - net animation	client-server (TCP/IP)	FS + DB
MVP-S	- declarative language instantiated via plan - entry/exit criteria	- role-based views of activities - status of activities	multi-client single-server (TCP/IP)	FS
Oikos	- logic concurrent language	- role-pad - object-orientation	client server in-house BMS	FS + DB
Oz	- rules - envelopes	- menu oriented - activity based - animation of task activation	multi-client multi-server (with TCP/IP)	homogeneous multi-db
Synervision	- shell-like language with process functions	- agenda (task) based (user and procedure)	peer-peer BMS	FS w/ locks and notification

2.1 HAKONIWA: A Process Monitor and Navigation System

The Hakoniwa system [3] is a project monitoring and navigation system which supports cooperative software development performed by a group of developers.

The system is based on a concurrent process model, which is composed of a set of tasks associated with communication primitives among the tasks. A task is defined as a sequence of primitive activities.

The Hakoniwa system is composed of two main components: i) a process monitor (Hakoniwa server); and ii) a process navigation system (task driver and organizer). Based on the assignment of tasks to each developer (which may be done by a project manager), task organizers for each developer and task drivers for each task are generated.

Major features of the Hakoniwa system are as follows:

- Activity navigation. A task organizer controls the task drivers which are associated with a developer. A task driver supports developers by providing menu selections for the next activities. These menus are automatically generated from the definition of the activity sequence.

  If an activity in the sequence is one accomplished by a tool invocation, the task driver automatically activates the tool.

- Progress monitoring. Each task driver reports to the Hakoniwa server log information of the task progress collected from the menu selection history. Thus, the project manager can capture the current status of whole project through the Hakoniwa server. The system displays the status of each task, and it also shows the history of activities for each developer.

- Communication support. All communication among tasks pass through the Hakoniwa server. Simple communication primitives such as task initiation request and task termination notification are automatically executed without any action of the developers.

At the start point of the project, the manager assigns some of project members to the initial tasks. Initial assignment is documented as a Hakoniwa task assignment file. Once the project starts, initial tasks are executed by members, and other succeeding tasks are predefined but not activated. They are instantiated/executed by assigning their enactor (member) on demand.

The focus of the demonstration included:

- global vs individual support;
- visualization of process execution;
- color based on the state of tasks;

- automatic generation of menus (from activity definition) guiding developer through activities;
- log of information collected (in server) from menu selections
- task assignment to enactor either pre-defined or assigned during execution.
- communication between tasks via messages.

2.2 LION Engineering Environment (LEU)

The LION Engineering Environment (LEU) [4] is a workflow management environment, which supports: data modeling (based on extended entity/relationship), process modeling (based on FUNSOFT nets) and organization modeling (based on organization charts), simulation, statistical analysis, and process enaction.

The Leu approach to software process modeling considers data models (describing object types and their relations), activity models (describing the activities to be carried out in a software process), and organization models (describing involved organizational entities and their roles) as separate, but equally important, facets of software processes.

Besides the tools to define the above mentioned models, specific features of Leu are demonstrated; for example, the generation of a database schema for a software project and the generation of standard dialogs used to insert or retrieve software artifacts to/from the project's database. Furthermore, Leu tools are demonstrated to model dialogs used by the software engineers to describe the states of tasks or to plan future activities.

The execution of software processes is based on the execution of the models mentioned above. The only interface between Leu and a software engineer is the "agenda". The agenda contains at any point in time all activities in all processes the software engineer can participate in. The calculation of an engineer's agenda is based on the engineer's access rights/roles and the states of currently executed processes. If an activity is started, the dialog, external tool, database query or batch function associated with the activity is executed. If an activity was completed, the agendas of all software engineers currently logged in are recalculated and adjusted to the changed process state.

To enable software engineers to overview a project's state, a process monitor provides information about all currently executed activities or the fulfilled and missing pre-conditions of activities. Besides monitoring currently running processes, simulation and analysis tools are used to analyze the future behavior of a process. For instance, bottlenecks and critical paths can be identified, potential process states can be simulated and the influence of resource changes on a project's execution plan can be tested.

LEU is a commercial product, a commercial reimplementation and enhancement of the FUNSOFT net approach. The effort to build such system has been 100 person/year since 1991. It is currently applied to software and other business processes, including housing building and administration. Leu is running in a client-server environment. The server needs to be a UNIX machine executing the software process models and controlling the database access. The agendas and dialogs are represented on the client which might be a unix-machine or even a PC. If available on the client's site, parts of an activity initiated through the agenda might be executed on the client too.

The focus of the demonstration included:

- Modeling different aspects of the process, i.e., data model, organization model and process model, and the integration of those aspects.
- Generation of standard dialogues (i.e., user interfaces out of object types) and their binding to the process.
- Analysis of the process, e.g., critical path analysis.

2.3 MVP-S: Support for Measurement-Based Project Guidance

The Multi-View Process System (MVP-S) [5] provides role-specific guidance to software developers during their projects based on explicit project plans, role definitions, quality models, and collected measurement data.

Major features of the system include:

- Process engine. This system accepts a textual MVP-L project plan and all related models, creates an internal representation of the plan, handles requests to query and manipulate the current project state, and maintains the project state across shutdowns of the host computer. It uses the information channel provided by the user interface to communicate with developers.
- User interface. People who play technical roles see a view of the project based on a "role-specific work context", which is the interface between

the developer and the set of activities (processes) which involve that developer. Multiple work context windows let developers view information about all of their activities. An activity-specific work context offers detailed guidance about each activity using quantitative quality models.

- Interaction model. Activities are coordinated via status values. The status values of activities shown in the role-specific work context window take on the values: i) disabled, i.e., can't be performed because entry criteria are false; ii) enabled, i.e., can be performed since entry criteria are true but no one is executing it; or iii) active, i.e., being performed. Upon receiving a request to start or complete an activity, the process engine checks the entry and exit criteria specified for the process and informs the requestor whether the request conforms to the project plan. Also, people interpret scripts.

- Use of empirical data. The tasks of collecting data are split between the user interface and the process engine. Tools are invoked by the user interface because the process engine does not necessarily have access to the work products on the user's machine. The process engine requests data directly from people by sending electronic mail. Empirical data may be used in the criteria to provide guidance by comparing actual values with target values. This use of measurement data goes beyond the approach of just signaling that a deviation from the project plan has been detected.

Measurement was integrated into the scenario to demonstrate the system's capabilities for guiding people according to role definitions and measurement data. For example, when completing a process step, the system will recognize a trigger condition, call a measurement tool to assess a product (automatic data collection), and request from the user the effort spent on the process (manual data collection). Further enaction of the project plan will be guided according to the resulting data. This can be used to coordinate steps within a role (guidance for a single person, or for all the people who play that role), as well as to coordinate steps between different roles (again, either an individual or multiple persons). This coordination helps people who are assigned tasks know what is expected of them, and assists people who play observational roles understand the current status of the project. Successive changes in the project state are communicated to all individuals who play roles within that project.

The focus of the demonstration included:

- explicit representation of processes and measurement;
- role-based views of activities in different states;
- collection of data automatically or user-directed;
- use of data for guidance and quality assurance.

2.4 OIKOS

The Oikos system [6] is based on a few principles:

- An enactable software process model, which consists of a hierarchy of interacting reactive systems including human actors. In fact, actors' roles are leaves in the hierarchy.
- The systems in a model are called "entities" and belong to different classes; each class embodies an important and well identified modeling concept, e.g., operating environment, managing site, actor's role.
- Service customization is an essential part of the model; thus, customizable pre-defined services provide the basic functionalities that support process enactment by accessing the allocated resources.
- An enactable model is developed by step-wise refinements. That means that several partial views of the process at various levels of abstraction can be extracted from the refinement structure.

Two languages are used in Oikos: a specification language, Limbo, and an enactment language, Paté, which is in fact an executable sub-language of Limbo. Limbo and Paté are concurrent logic languages. Their basic features are: a) entities are organized in a tree, reflecting the final structure of the enactable model; b) agents are defined as sets of nondeterministic reaction rules, of the kind condition-computation-action; rules in a set can be partially serialized by path-expressions; and c) agents in an entity evaluate concurrently their rules, according to the blackboard model, i.e. they share a common associative memory.

Expo is the Oikos run-time support. The main goals of Expo are to provide distributed execution of Paté systems, integration of off-the-shelf tools and non Paté systems, and multiple human interaction with Paté systems. A compiler translates a Paté program into an intermediate code, which is interpreted by a collection of predefined BIMprolog processes. The user interface is based on OSF/Motif. Interprocess communication is achieved through Unix sockets.

Persistency is introduced by the MCC logical DBMS Salad.

For the demonstration, the scenario was extended with the PSS05 Standard of the European Space Agency; i.e., a three-stage PSS05 procedure for local change. The following suggested sub-scenarios were also included:

- Specific procedures/policies: PSS05, configuration control and versioning capabilities.
- User role support. Roles are among the basic modeling concepts of Oikos, which allows one to express coordination of the roles in the process with respect to the available resources. Static assignment of a single (Unix) user to multiple roles is supported.
- Individual Support is provided via "RolePads", which guide users who play the role along the appropriate behavioral pattern. Thus, at any given moment it shows the available documents and the actions that the user can perform on them, i.e. which tools can be invoked on which documents and which coordination actions are available.

The focus of the demonstration included:

- interactive reactive system
- user guidance via "rolepad" user interaction
- enactment and simulation of concurrent executing roles
- object orientation via icons representing object and menus representing actions
- addition of PSS05 procedure.

2.5 Oz: A Decentralized Process Centered Environment

Oz [7] is a Process Centered Environment that supports — in addition to modeling and enactment of a (single- and multi-user) project-specific process — modeling and enactment of multiple heterogeneous, autonomous, and possibly physically dispersed, processes.

This research project focuses on two main aspects:

- Process interoperability, i.e., to investigate the extension of the concepts of modeling and enactment to assist teams (or individuals), each with its own process, to define and execute collaborative activities while still retaining the desired privacy of each team's process; and
- Process interconnectivity, i.e., to investigate the architectural infrastructure that is required to support process interoperability.

Oz is the successor of the Marvel project. As such, it employs similar formalisms and mechanisms supporting a single process. Specifically, it uses object-oriented data modeling, rule-based process modeling, and a client-server architecture with a reactive server that manages the process, object base and intra-process coordination of an instantiated environment. The client provides a graphical user-interface for browsing and querying the objectbase and the process definition and a mechanism for executing enveloped activities.

Process interoperability in Oz is provided by the Treaty formalism for modeling collaboration among processes which are by default private, and the Summit mechanism for enacting the defined Treaties. The basic idea behind the Treaty is to enable the dynamic definition (and retraction) of shared sub-processes as extensions to the (possibly pre-existing) private processes. In addition, the Treaty mechanism requires the involved parties to actively participate and agree on the Treaty's contents. The gist of the Summit is to enable execution of Treaty sub-processes but retain the locality of non-shared processes.

Process interconnectivity is supported in Oz by a multi-server architecture, where each server corresponds to a (software) process, and by a semi-replicated connection database that is itself maintained and manipulated by a (configuration) process. The main emphasis here is on the "shared-nothing" property that enables independent and self-contained operation of the possibly geographically dispersed groups while still supporting maximum process interconnectivity on demand.

The demonstration had as objective to show how the process formalism and its underlying execution engine can be used to assist (teams of) users in performing their tasks, and how human-oriented activities and tools are integrated into the process. It expanded the base scenario to show key features of Oz:

- Several interacting processes (e.g., one for the testing group and two separate development processes). The demonstration shows how potential interactions can be added (removed) on the fly, and how entire (sub)-environments can be added to and removed from a global environment using the registration process. (e.g., process changes while in execution)
- Disjoint and asynchronous individual work in

the local processes versus joint and synchronous project work.

- Process support for modeling and enacting user-delegation, with emphasis on dynamic user binding (which may or may not be tied to the notion of user roles).
- Process support for modeling, enacting, and integrating multi-user tools, including both in-house and off-the-shelf tools. Also shows integration with a multi-user collaboration tool and a configuration management tool.
- Automatic enactment of the process and use of the project database to carry out some process steps (e.g., approval/rejection)

The focus of the demonstration included:

- support for multiple cooperating processes (maximizing autonomy), showing interoperability, heterogeneity and decentralization;
- support for multiple and cooperating users (CSCW) via: delegation, and synchronization of multi-user tools including "white board" and "audio/editing".

2.6 SynerVision

SynerVision [8] is a commercial process enactment tool. The enactment model is defined in SynerVision and used by SynerVision. It has the following characteristics:

- The representation model is a hierarchy of tasks (a work breakdown structure) having attributes and whose visibility is shared by a work group. All users' tasks (not just ones related to a specific process instance or class) are represented.
- Humans in the work group apply filters to sort and filter the tasks in a way that helps them focus on tasks of concern.
- Attributes of the tasks are used for the following purposes: representing task state, relationships, and constraints, associating automations with the task, and providing task guidance. User defined attributes may be added to the schema to represent any of the above or additional information.
- Attributes provided by and used by the base tool include: automatic actions which are invoked upon certain state transitions or which determine whether certain transitions are allowed; manual actions which are helpful in completing the task but are invoked at the request of the user only; dependencies between tasks; status of the task (completed, abandoned, in progress, new or ongoing); owner of the task; whether the task is currently being executed; notes for both process guidance and to record information about the task; and others.
- Humans and automated agents are modeled as owners of tasks. Tools which are not agents to which tasks are assigned, but merely contribute to the implementation of some actions attached to tasks are not modeled per se – they are simply invoked within the defined action attached to the task, either via the SoftBench Broadcast Message Server (BMS) or via Unix commands.

The demonstration included the environment ChangeVision. ChangeVision is an environment built on top of SynerVision which provides a change request process (process templates and tools which interact with the process defined by the templates). Each change request in the system is an instance of the change request process and has the standard actions, subtasks, dependencies, etc, associated with it. ChangeVision integrates a configuration management, version control system and a defect tracking system, together with SynerVision; it also includes some metrics utilities. The tools provided/integrated are those that are useful to the change request process.

The focus of the demonstration included:

- automation of lots of mundane tasks;
- showing the intermixing of process and user generated tasks;
- people coordination via delegation and dependencies between users' tasks;
- visibility of all tasks by everyone
- filtering capability associated with tasks
- configuration management, on work done by user in process.

2.7 SPADE (not demonstrated)

The SPADE [9] project goal is to provide a software engineering environment to support Software Process

Analysis, Design, and Enactment. The project is currently being carried out at CEFRIEL and Politecnico di Milano. The environment is based on a process modeling language, called SLANG (SPADE Language), which is a high-level Petri net based formalism. SLANG offers features for process modeling, enactment, and evolution. In addition, it describes interaction with external tools and humans in a uniform manner. The main features of the SLANG modeling facilities can be summarized as follows:

o Process models can be statically structured in a modular way using the activity construct. Activities (i.e., process fragments) can be dynamically instantiated.

o Activities can be manipulated as data by other activities; i.e., SLANG supports computational reflection.

o Process artifacts, including process models, are modeled as tokens of a Petri net and behave as instances of abstract data types (i.e., in an object-oriented manner).

SPADE-1 is the first implementation of the SPADE environment; it supports the enactment of SLANG process models. Moreover, it provides the basic mechanisms for process model evolution. Thanks to SLANG reflective features, a SLANG process model may include a metaprocess to change process definition and/or state during process model enactment. SPADE-1 architecture is based on the principle of separation of concerns between process model enactment and user interaction. This means that the semantics of the process modeling language does not preclude any user interaction paradigm, achieving independence of user interaction paradigm and process modeling paradigm.

The implementation of the SPADE-1 repository is based on the object-oriented database management system O2. The process model and the process data are both stored in the repository. Software artifacts manipulated by the process model (tokens) correspond to O2 objects.

2.8 Regatta Technology Tool (not demonstrated)

Regatta Technology is a commercial visual business process modeling and enactment tool. Wide flexibility is achieved by allowing end users to design and actively modify their own process plans through an easy-to-use Petri-net based graphical representation on Motif, and MS windows. Cross platform API extends support to custom applications.

The essential unique feature of the Regatta Technology is: firstly, that it is designed for end-users to create and modify process descriptions through an easy to use graphical description; and secondly, that it allows descriptions of the process to be modified at any time, even while the process is being enacted.

3 Commentary

The demonstration day was a great success at encouraging in-depth discussions about interaction paradigms and architectures of PSEEs; those discussions were significantly strengthened by the concrete views of the user interfaces and interactions of the demonstrated systems. Many feel that interspersing demonstrations with the workshop sessions will enrich the discussions and provide more concrete data for discussions.

Since the audience consisted of mostly PSEE builders, there was wide interest in understanding the architectures of such systems (not really obvious during the demonstrations). It is felt that a lot more discussion and understanding is needed about how those systems are constructed, how the architecture of systems support the specific process modeling and enactment techniques, and what are the relationships among architectures, run-time support for process enactment, user interaction paradigms, and the various characteristics demonstrated. At the suggestion of the attendees, we collected architecture depictions of the various systems and included them in the appendix of this document.

It is worth pointing out that two commercial and four research systems were demonstrated. The amount of effort for putting together the demonstration varied among the various systems. The demonstrations covered the basic scenario and many dealt with aspects of the sub-scenarios. A few comments about similar concepts among the demonstrated systems are: a) individual support is necessary beyond global, central project support; b) agendas or role-pads for individual support are being used and are recommended; c) the formalisms did not seem to have an impact on the choice of user interaction paradigms; d) the "Process Engine" appears as a key component in those systems' architectures; e) the interoperability mechanism varies but it seems that message-based systems are gaining strength; f) triggering mechanisms are in wide use for communication among processes, and among processes and data.

4 Acknowledgement.

The coordination of this activity was supported by the Advanced Research Projects Agency, under contract N00039-95-C-0017, issued by the Space and Naval Warfare Systems Command.

5 Bibliography

1. Penedo, Maria H., "Life-cycle (Sub) Process Scenario for 9th International Software Process Workshop (ISPW9)", *Proceedings of the 9th International Software Process Workshop*, Arlie, VA, October 1994.

2. Kellner, M., P. Feiler, A. Finkelstein, T. Katayama, L. Osterweil, M. Penedo, D. Rombach, "ISPW-6 Software Process Example", *Proceedings of the 1st International Conference on the Software Process*, California, October 1991.

3. Hakoniwa System:

   • Iida, H., Mimura, K., Inoue, K. and Torii, K., "Hakoniwa: monitor and navigation system for cooperative development based on activity sequence model," in *Proceedings of 2nd ICSP*, pp.64-74, February 1993.

4. LEU System:

   • G. Dinkhoff, V. Gruhn, A. Saalmann and M. Zielonka, Business Process Modeling in the Workflow Management Environment LEU, *Proceedings of the 13th International Conference on the Entity-Relationship Approach*, Manchester, UK, December, 1994.

   • V. Gruhn, "Communication Support in the Workflow Management Environment LEU", *Connectivity '94 - Workflow Management - Challenges, Paradigms and Products*, Linz, Austria, R. Oldenbourg, Vienna, Munich, G. Chroust, A. Benczur (eds.), pages 187-200, October, 1994.

5. MVP-S System:

   • Christopher M. Lott, "Measurement support in software engineering environments," *International Journal of Software Engineering & Knowledge Engineering*, 4(3), September 1994.

   • Christopher M. Lott, Barbara Hoisl and H. Dieter Rombach, "The use of roles and measurement to enact project plans in MVP-S", to appear in *Proceedings of the 4th European Workshop on Software Process Technology*, April 1995.

6. OIKOS System:

   • C. Montangero and V. Ambriola, "Oikos: Constructing Process-centred SDEs", in *A. Finkelstein, J. Kramer and B. Nuseibeh (eds)*, "Software Process Modelling and Technology", Research Study Press, Taunton, 1994, 131-151.

   • V. Ambriola, G.A.Cignoni and C. Montangero, "The Oikos Services for Object Management in the Software Process", in *B.C. Warboys (ed) Software Process Technology*, EWSPT'94, Feb 94, LNCS 772, 2-14.

7. OZ System:

   • Israel Z. Ben-Shaul and Gail E. Kaiser, "A Paradigm for Decentralized Process Modeling and its Realization in the Oz Environment", *Proceedings of 16th International Conference on Software Engineering*, IEEE Computer Society Press, pp. 179-188, Sorrento, Italy, May 1994.

   • Israel Z. Ben-Shaul and Gail E. Kaiser, "A Configuration Process for a Distributed Software Development Environment", *2nd International Workshop on Configurable Distributed Systems*, pp. 123-134, Pittsburgh PA, March 1994.

8. Synervision:

   • B. Fromme and J. Walker, "An Open Architecture for Tool and Process Integration," *Proceedings of the Software Engineering Environments Conference*, pp. 50-62, IEEE Computer Society Press, July 7-9, 1993.

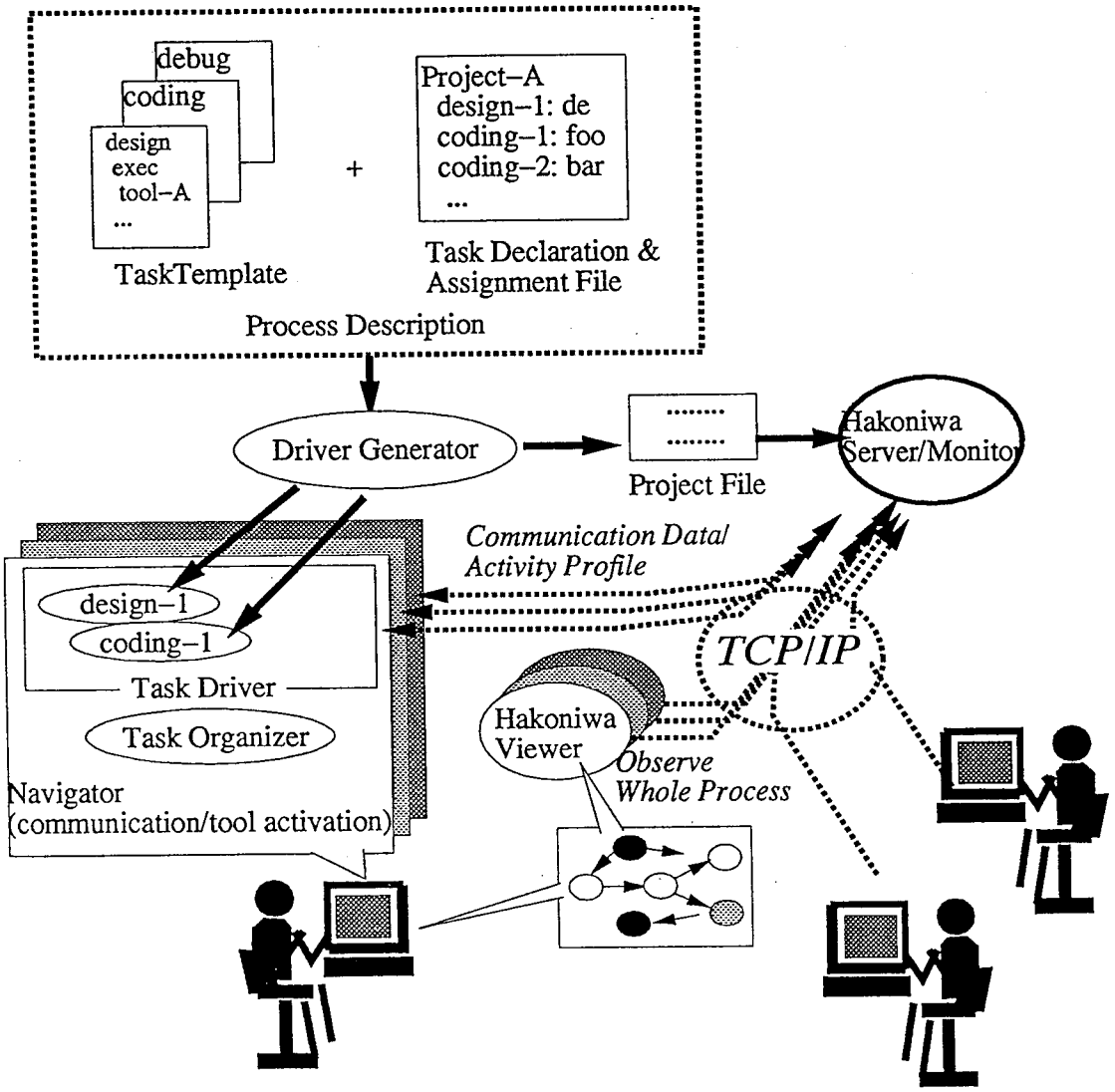
   • J. Diamant, "Human Interaction Support in HP SynerVision for SoftBench" *Proceedings of the 9th International Software Process Workshop*, IEEE Computer Society Press, Arlie, October 1994.

9. Spade:

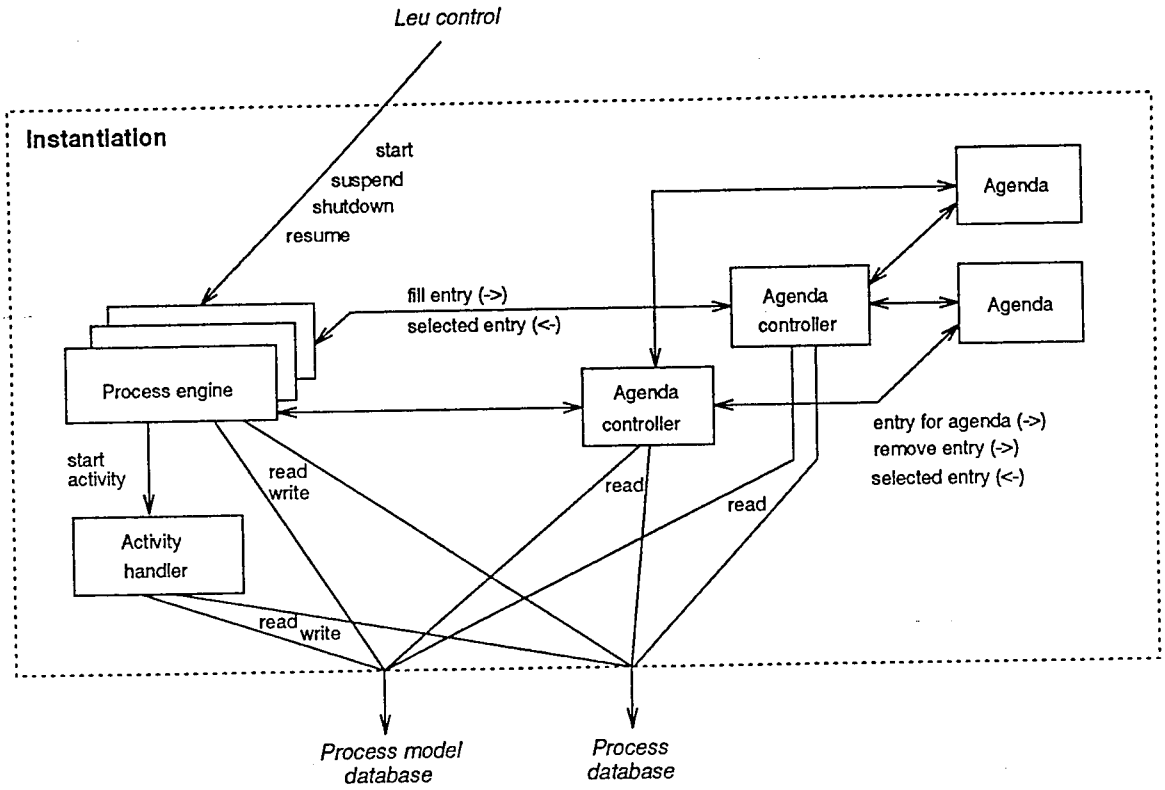
   • Sergio Bandinelli, Alfonso Fuggetta, and Carlo Ghezzi, "Process Model Evolution in the SPADE Environment", *IEEE Transactions on Software Engineering*, 19(12):1128–1144, December 1993.

   • Sergio Bandinelli, Alfonso Fuggetta, Carlo Ghezzi, and Luigi Lavazza, "SPADE: An Environment for Software Process Analysis, Design and Enactment", in *Anthony Finkelstein, Jeff Kramer, and Bashar Nuseibeh, editors, Software Process Modelling and Technology*, pages 223–247. Research Studies Press Limited, 1994.

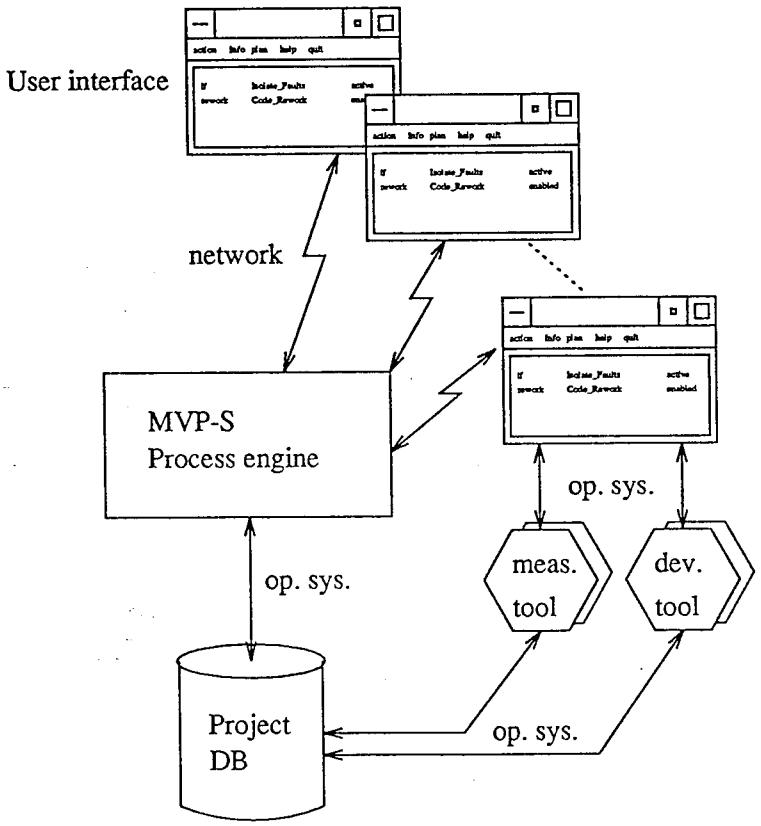
Appendix - Architectures of the Demonstrated Systems.



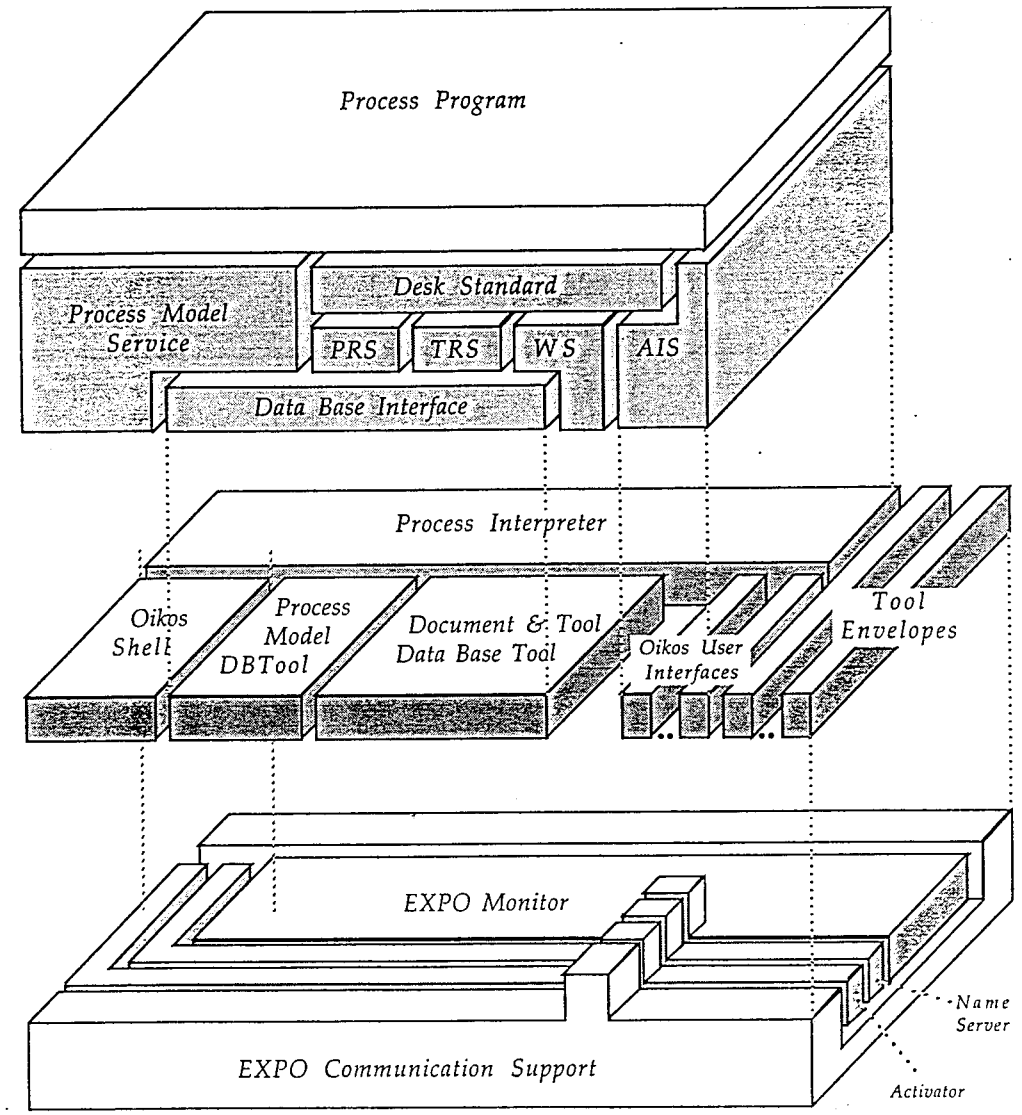
A View of the Hakoniwa System Architecture



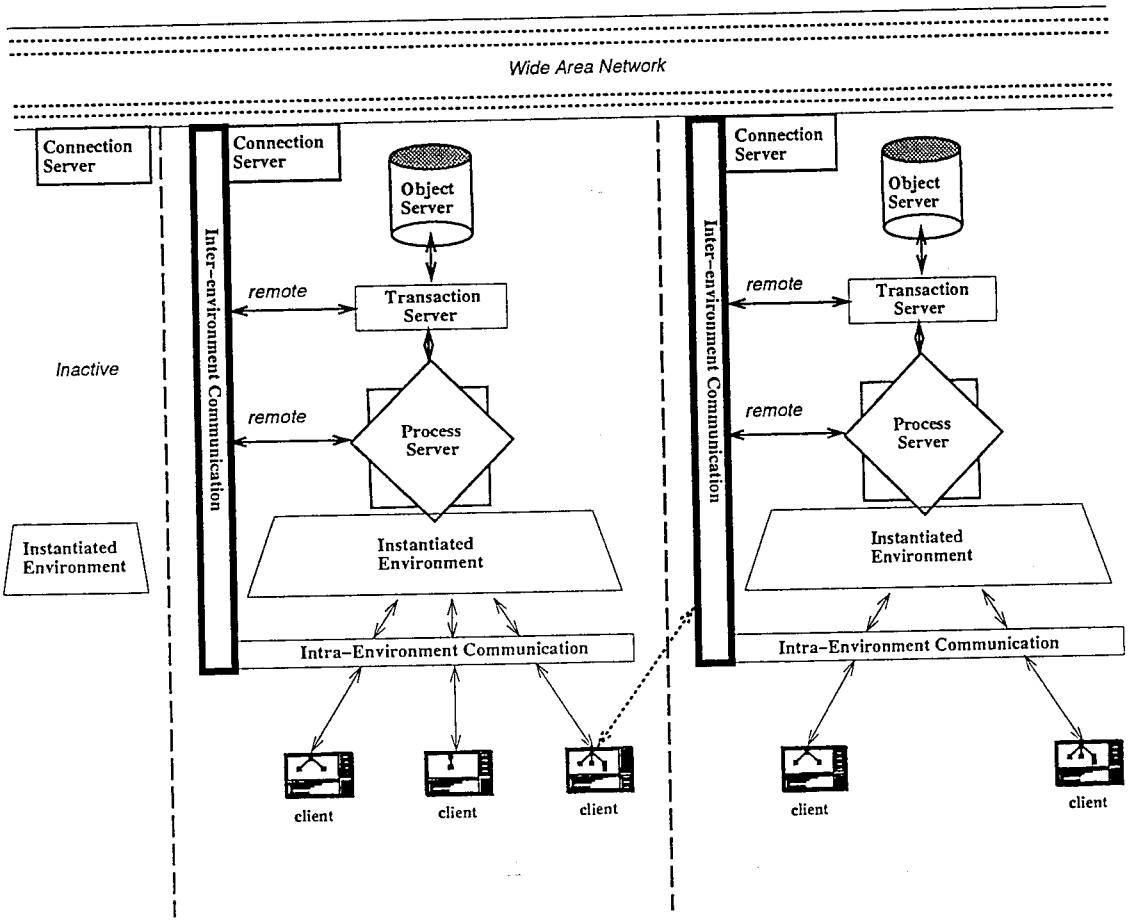
A View of the Leu System Architecture



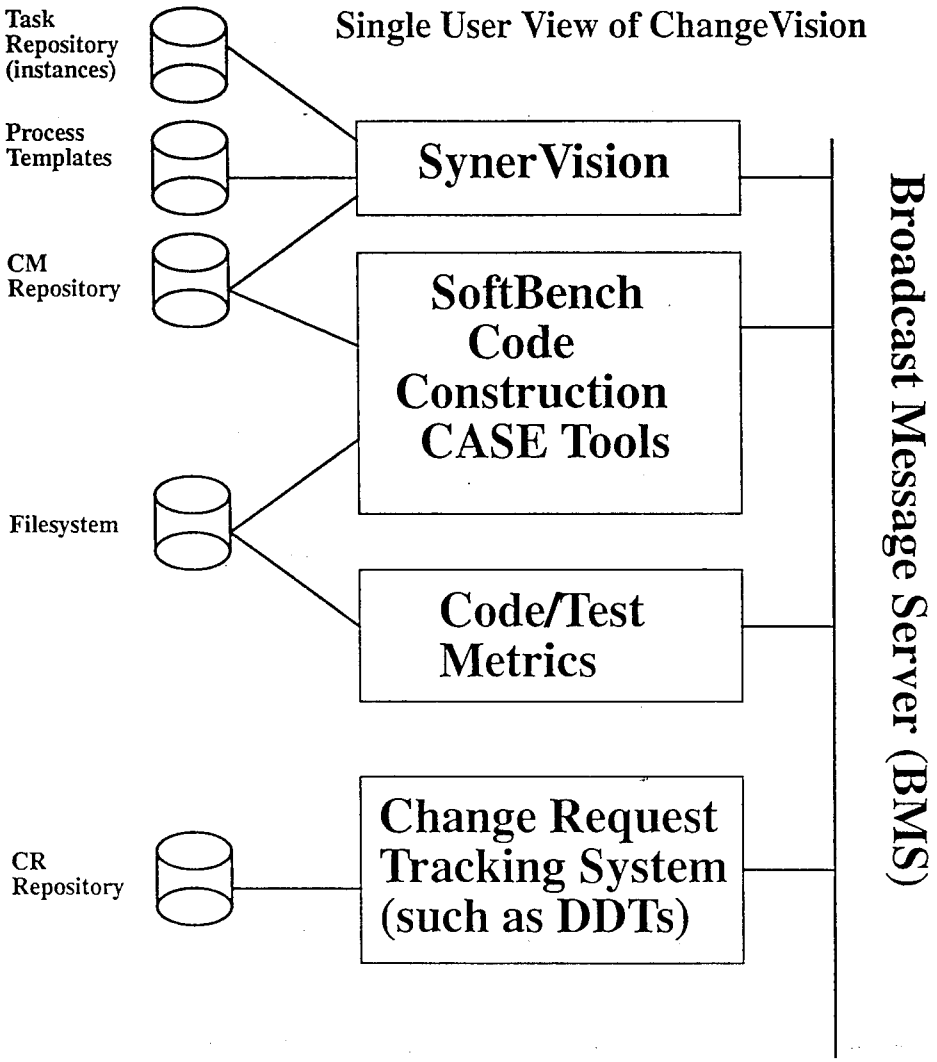
A View of the MVP-S System Architecture



A View of the Oikos System Architecture



A View of the Oz System Architecture



A View of the Synervision System Architecture

SBUS: A Framework for Software Bus Comparison*

Maria H. Penedo
Christine Shu
TRW
One Space Park
Redondo Beach, CA 90278

Abstract

This paper outlines an initial framework, denoted SBUS, for the characterization and comparison of systems or mechanisms which are identified as software buses. "Software Buses" play an important role in supporting component interoperability in Software Engineering Environment (SEE) architectures. The SBUS framework consists of a set of attributes which together characterize such systems. An initial survey based on this framework appears in [1]. The systems surveyed were: HP's BMS, Forest, ESF K/1's Software Bus, ESF Kernel/2r's Muse, Polylith, Arcadia's Q, Weaves. This paper outlines the SBUS attributes and characteristics and illustrates its use by characterizing aspects of the Arcadia's Q system. Both the framework and the survey represent work in progress.

1 Background of Work

Over the last decade, part of the software engineering community has been shifting its attention towards software engineering environment (SEE) and process issues, finding that tools and languages are best defined and implemented within the context of environments in which they are used and the processes they support. Key requirements for next generation SEEs which have direct impact on their architecture, i.e, on the way they are built, are: component-based and interoperability technology, rapid construction and adaptation technology, extensibility, and support for (life-cycle) process automation.

Our research activities have aimed at providing solutions in support of those requirements. A major objective of our approach is to create, assess and enhance the technology necessary *to rapidly build and sustain pro-active, component-based and process-driven SEEs (PSEE)*. Towards this goal we are defining and validating a Domain Specific Software Architecture (DSSA) approach to PSEEs. We have been exploring and prototyping architectural issues from the perspectives of interfaces and integration [PS91, PSSS89] and defining models to serve as functional reference frameworks or architectures [NIS93, KPS93, Pen93] for PSEEs.

A recent workshop on Process-sensitive Software Engineering Environment Architecture[1] indicated that further technology in support of componentization of PSEE components and process components is needed. It assessed the state of the art, it brought up important architectural issues and made recommendations for future work [PR93]. At that workshop, consensus was achieved on the following definition: *"A software architecture should be viewed and described from different perspectives and it should identify:*

(a) its components,

(b) their static inter-relationships

(c) their dynamic interactions

(d) properties and characteristics

(e) constraints on the items above."

*submitted to ICSE-17 Workshop on Architectures for Software Systems.

[1]This workshop was coordinated by us in cooperation with the Rocky Mountain Institute of Software Engineering.

SEE architectures are very important in the realm of software architecture studies. SEEs are systems in their own right, thus representing a specific system/software domain, and SEEs may include components in support of software architecture design and implementation.

2 SBUS Introduction

The term "*software bus*" has been frequently used in recent years for describing a class of architectures that share a common component (denoted *software bus*) which implements an abstract communications model for the interoperation of components in a distributed environment. There are some essential characteristics and behavior of these systems that delineate them from other architectural approaches. The PSEEA workshop [PR93], for example, used three architectural approaches for characterizing existing PSEEs with respect to their component communication: i) logically centralized database, ii) direct agent to agent (connections), and iii) software bus.

Software buses are currently being used as key mechanisms for SEE tool/component communication and interoperation. We strongly believe that a better understanding of interoperability mechanisms, characterizing the circumstances under which one is preferred to the other is necessary to support the rapid construction of SEEs. Towards this goal, we have identified a reference framework, denoted SBUS (Software Bus), in order to better understand and compare software bus components. We note that a software bus system can be composed of one or more software components (which themselves may communicate via their own bus mechanism).

SBUS consists of a set of attributes/characteristics which are applicable towards describing software buses. This set of attributes is evolving as our investigations proceed and our knowledge increases. We performed a survey of systems which have been characterized as software buses and play an important role in existing Software Engineering Environment (SEE) architectures. The systems surveyed are: HP's BMS, Forest, ESF K/1's Software Bus, ESF Kernel/2r's Muse, Polylith, Arcadia's Q, Weaves. Further details on the application of such framework to existing systems can be found in [SP93].

This paper outlines the SBUS attributes and characteristics and illustrates its use by characterizing aspects of the Arcadia's Q system [Hei92, MH+92, May92][2]. Q is a key interoperability component in the Arcadia set of environment components; it is an enhanced remote-procedure-call component which allows a client to invoke an arbitrary server dynamically. Q has been extensively used in the integration of Arcadia SEE components.

3 SBUS Framework: Attributes and Characteristics

This section describes a Software BUS reference framework, denoted SBUS, for the description and comparison of SEE software bus systems. It consists of a set of attributes or characteristics. The objective of this framework is to help us understand the characteristics of such systems in order to define guidelines for using them in the construction of SEEs. As our investigations proceed, new attributes are being defined, and old ones are refined.

3.1 Primary Purpose

This attribute describes the primary purpose or objective of the software bus system.

For example, a key objective of the Q system is to support the interconnection of multi-lingual software components for execution in heterogeneous environments.

3.2 Communication Model

This attribute describes the kind of communication model supported by the software bus, i.e., how components communicate. There are many subcharacterizations of the ways components communicate, as described below.

1. *Direct vs. Mediated Communication.* This attribute describes whether the software bus provides a dedicated communication channel between a sender (of a message) and a receiver, or the message is intercepted and delivered by an external agent.

2. *Point-to-point vs. Broadcast vs. Multi-cast.* This attribute typically applies to message-based systems. These elements are as follows:

   - Point-to-point. The sender explicitly identifies the receiver.

[2] We note that the Q system may have evolved since our evaluation, which was based on the documentation available and discussions with its developers.

- Broadcast. The message is broadcast to every component that may be listening in the environment
- Multi-cast - There is some selection mechanism that identifies a subset of components to receive the message.

3. *Client/server or peer-to-peer.* This attribute describes whether, within a single thread of execution, the communication between two components is client/server or peer-to-peer.

   Client/server is an environment where the requester (client) of a service is on one system and the supplier (server) of the service is potentially on another system. Their arity may vary:

   - multi-client, where more than one client makes requests of the server;
   - multi-server, where one client makes multiple requests to multiple servers; and
   - multiple client and multiple server.

   Peer-to-peer is an environment where two clients (potentially on separate processors) can submit requests to one another over a single logical connection - that is, a single communications sequence over the network. Each client in this case can also be a server.

4. *Location Transparency.* This attribute describes whether a component needs to know the location of the component with which it communicates.

5. *Naming.* This attribute describes whether the components for communication need to be addressed explicitly. There are cases where abstract names can be used.

Example: The Q system supports the following communication model:

1. Direct communication
2. Point-to-point communication
3. Client/server system, supporting multi-client, multi-server, or multi-client/server.
4. Location. Client is aware of the location of the server.
5. Naming. Explicit addressing.

3.3 Run-time Behavior

This attribute describes the run-time characteristics supported by the software bus.

In the Q system, the client issues a service request by sending a message to the server process. The client can continue processing, if appropriate. The server remains inactive until it is awakened through a signal based notification mechanism to process the request.

3.4 Bus Interface Description

This attribute describes how components use or interface with the bus, e.g., via message-based, procedure-call-based semantics, or some variant. It also describes the interface and its parameters.

Example: Q supports a procedure call interface. The client specifies: server (machine, server id, version), service type, and Q-data representation (QDR) buffer.

3.5 Binding Time

This attribute describes the binding time. "Early binding" implies static/compile time association whereas "late binding" implies run-time association. However, there can be several levels of binding. For example: i) When is the association between an abstract service name and the actual physical software that implements the service made? ii) When is the association between the service requester and the service provider made? iii) Does the service requester have to know apriori what services exist in the environment? iv) Can services be dynamically added to an environment and be accessed by other components in a non-intrusive way (without recompilation or relinking of existing components)?

Example: Q supports late binding.

3.6 Data Granularity and Type

This attribute describes the granularity of the data being passed by the bus.

Example: Q supports both primitive types (i.e., integer, string, boolean, and float) and composite types that are based on combination of primitive types using vectors and record structures.

3.7 Process Granularity

This attribute describes the granularity of the process elements being communicated (e.g., routine, procedure, operating system process, Ada task).

Example: Q supports communication between UNIX processes.

3.8 Multilingual Support

This attribute describes whether the bus allows communication between components written in different languages, and, if so, what kind of language bindings are provided.

Example: Q supports multiple languages, C, Ada, C++, E, LISP, and Prolog.

3.9 Data Translation

This attribute describes whether data translation is supported and how transparent this translation is to the communicating components.

Example: In Q, data translation is not transparent to the application. The Q client explicitly encodes its arguments prior to issuing a remote procedure call (RPC) and explicitly decodes any returned results.

3.10 Protocols

This attribute describes the underlying protocols used to implement the bus. It may describe whether the message content at different levels of abstraction conform with some standard protocol.

Example: Q is implemented on top of modified RPC (ARPC). It also depends on Sockets, TCP/IP, UDP. It uses a data representation protocol (QDR).

3.11 I/O Synchronization

A bus interface can be synchronous, asynchronous or both, as follows:

- Synchronous protocol - a communications protocol in which the component acting as client suspends execution of its current process until it receives a response from either the component acting as server or the software bus.
- Asynchronous protocol - a communications protocol in which the component acting as client does not wait for a response from the component acting as server even though one may be expected in due course (this implies that responses are handled either by polling or interrupts).

This attribute should also specify whether the requesting component blocks when communicating with other component via the bus.

Example: Q supports synchronous and signal-based asynchronous communication. In signal-based asynchronous communication, the sockets are configured for asynchronous I/O. When a service request arrives at the socket, a signal is sent to the service dispatcher. The service dispatcher will call the appropriate service procedure to furnish the service requested. The service procedure may "acknowledge" completion of the service allowing the client to continue execution concurrently with the service procedure.

3.12 Triggering

This attribute describes whether the software bus provides the capability to intercept messages on the bus and trigger actions based on those messages.

Example: In Q, there is no support for triggering.

3.13 Threads

Threading refers to the ability to divide a program into multiple parts that execute concurrently within the same virtual address space. A multi-threaded program has multiple points of execution interleaving faster computational operations with slower operations. This attribute describes whether the bus is single or multi-threaded. Other applicable questions may deal with whether the server is re-entrant.

Example: Q is single threaded for C and multi-threaded for Ada. It was designed to support concurrently active communicating servers.

3.14 Scope

This attribute describes the scope of the bus, i.e., whether the bus is used for coordinating communication of components: a) within a single process family; b) across multiple process families within the same processor; or c) across multiple process families and across multiple processors.

3.15 Distribution

This attribute describes whether the bus supports distribution across multiple processors and whether this distribution is transparent to the components. It should also describe how distribution is specified and whether it is static or dynamic.

Example: Q supports distribution but it is not transparent to application.

3.16 Component Interface Specification

This attribute describes whether there is linguistic support for specifying component interfaces. It should also describe whether it supports the generation of interface stubs.

3.17 Registration

This attribute describes how components are registered in the software bus environment. It should include information such as: a) whether the registration of components is static or dynamic; b) how components are changed or replaced; c) whether there is support for dynamic update of communicating components.

3.18 Exception Handling

This attribute describes how exceptions are generated, captured, handled, and/or propagated.

3.19 Security

This attribute describes the security model supported by the bus, if any. For example, whether there are access control mechanisms provided to ensure varying degrees of secure communication.

3.20 Versioning

This attribute describes any versioning support.

3.21 Software Bus Development Tools

This attribute describes the software bus tools (e.g., development, generative, analysis, browsing) which support the development of software bus applications.

3.22 Platform Dependencies

This attribute describes any hardware and/or software dependencies.

Example: Q is built on SUN XDR/RPC.

3.23 Strengths

This attribute describes the primary strengths of the architecture supported by this software bus.

3.24 Weaknesses

This attribute describes the main limitations of this architecture, e.g., hardware, software, language, error detection/correction, functionality, etc.

4 Conclusions.

In this paper we described the SBUS framework, a framework for the understanding and comparison of systems considered as SEE software buses. The SBUS framework consists of a set of attributes and characteristics. A survey of such systems using the framework has been done and it appears in [SP93] This framework has benefitted (i.e., was enhanced) as a result of studying those system's descriptions. In this paper, we exemplify the use of the framework by characterizing the Arcadia Q system.

It is worth noting that our objective is **not** to compare whether a software bus system is "better" than the other; the objective of the SBUS framework is to help us understand the characteristics of such systems in order to define guidelines for using them in the construction of SEEs.

Other candidate systems mentioned in literature but not addressed here are: DCE, Ole2, SCORPION, CORBA, ToolTalk, Matchmaker/MIG, Mercury, Isis, SLI, Abe, Conic, Durra, Infuse, HPC/HRPC, Mercury, MLP. We plan to continue our work by characterizing those systems using the SBUS framework.

This document is the continuation of investigations towards understanding the complex issue of componentization and interconnectivity of heterogeneous components. Much more work lies ahead before we can fully understand when and how those components should be interconnected to fulfill specific software projects and domain requirements.

5 Acknowledgement.

We would like to acknowledge the interesting and fruitful discussions we have held with D. Heimbigner, S. Sutton and M. Maybee on this subject.

This work was supported by the Advanced Research Projects Agency, under contracts #N00039-91-C-0151 and N00039-95-C-0017, issued by the Space and Naval Warfare Systems Command.

References

[Hei92] D. H. Heimbigner. ARPC: An Augmented Remote Procedure Call System. Technical Report CU-Arcadia-100-92, Department of Computer Science, University of Colorado, October 19 1992.

[KPS93] A. Karrer, M. H. Penedo, and C. Shu. A Survey of Software Engineering Environment Architecture Approaches. Technical Report Arcadia-TRW-93-007, TRW, Redondo Beach, CA, 1990, November 1993.

[May92] M. J. Maybee. *Q: A Multi-lingual Interprocess Communications System - Reference Manual*, February 1992.

[MH+92] M. J. Maybee, D. H. Heimbigner, et al. Q: A Multi-lingual Interprocess Communications System for Software environment Implementation. Technical report, Department of Computer Science, University of Colorado, 1992.

[NIS93] Reference Model for Frameworks of Software Engineering Environments. Technical Report NIST Special Publication 500-211 and ECMA/TC33 Technical Report TR/55, 3rd Edition, August 1993.

[Pen93] M. H. Penedo. Towards understanding Software Engineering Environments. In *Proceedings of TRW Conference on Integrated Computer-Aided Software Engineering*, California, November 1993. also in TRW Technical Report IMPSEE-TRW-93-003.

[PR93] M. H. Penedo and W.E. Riddle. Process-sensitive Software Engineering Environment Architectures - Summary Report. In *ACM Software Engineering Notes*, July 1993.

[PS91] M.H. Penedo and C. Shu. Acquiring Experiences with the Modeling and Implementation of the Project Life-cycle Process - the PMDB work. *IEE and British Computer Society Software Engineering Journal*, September 1991.

[PSSS89] M.H. Penedo, C. Shu, S. Simpson, and S. Sykes. PMDB+ Viewer Architecture Report. Technical Report Arcadia-TRW-89-016, TRW, December 1989.

[SP93] C. Shu and M. H. Penedo. SEE Software Bus Survey. Technical Report IMPSEE-TRW-93-008, TRW, Redondo Beach, CA, December 1993.

SEE Software Bus Survey

TRW

December 1993

C. Shu
M. H. Penedo

TRW Technical Report IMPSEE-TRW-93-008

Abstract

This document presents an initial survey of systems which have been characterized as software buses and play an important role in tying together components in Software Engineering Environment (SEE) architectures. It documents work in progress. The survey has been based on a subset of the papers listed in the document; thus, it may be incomplete and inaccurate with respect to the current status of such systems. The systems surveyed are: HP's BMS, Forest, ESF K/1's Software Bus, ESF Kernel/2r's Muse, Polylith, Arcadia's Q, Weaves. Other candidate systems should be added in later versions of this document.

A framework consisting of attributes and characteristics was defined and used for describing those systems' characteristics.

Contents

1 Introduction **3**

2 SBUS Framework: Attributes/Characteristics **3**

2.1 Primary Purpose . . . 3
2.2 Communication Model . . . 3
2.3 Run-time Behavior . . . 4
2.4 Bus Interface Description . . . 4
2.5 Binding Time . . . 4
2.6 Data Granularity and Type . . . 5
2.7 Process Granularity . . . 5
2.8 Multilingual Support . . . 5
2.9 Data Translation . . . 5
2.10 Protocols . . . 5
2.11 I/O Synchronization . . . 5
2.12 Triggering . . . 5
2.13 Threads . . . 6
2.14 Scope . . . 6
2.15 Distribution . . . 6
2.16 Component Interface Specification . . . 6
2.17 Registration . . . 6
2.18 Exception Handling . . . 6
2.19 Security . . . 6
2.20 Versioning . . . 6
2.21 Software Bus Development Tools . . . 7
2.22 Platform Dependencies . . . 7
2.23 Strengths . . . 7
2.24 Weaknesses . . . 7

3 Systems Surveyed **7**

3.1 BMS . . . 7
3.2 Forest . . . 9
3.3 K/1 Software Bus. . . . 10
3.4 K/2r MUSE Software Bus. . . . 14
3.5 Polylith Software Bus. . . . 15
3.6 Q . . . 17
3.7 Weaves . . . 18

4 Conclusions. **20**

5 Acknowledgement. **20**

A Software Bus Survey Highlights. **23**

1 Introduction

The term "*software bus*" has emerged frequently in recent years for describing a class of architectures that share a common component (denoted a *software bus*) which implements an abstract communications model for the interoperation of components in a distributed environment. There are some essential characteristics and behavior of these systems that delineate them from other architectural approaches. This report documents work in progress and presents an initial survey of systems which have been characterized as software buses and play an important role in tying together components in Software Engineering Environment (SEE) architectures. Software bus components are currently being used as a key mechanism for SEE tool/component communication and interoperation.

In order to better understand and compare software bus components, we have identified a reference framework, i.e., a set of attributes/characteristics which are applicable towards describing such systems. This set is evolving as our investigations proceed and our knowledge increases. The attributes and characteristics are described in Section 2.

Section 3 presents a first draft of a survey of systems which have been characterized as software buses; it uses the framework described in Section 2. This survey is preliminary and it has been based on a subset of the documentation listed in the document, since some documents were obtained recently. Therefore, this survey may be incomplete and inaccurate with respect to the current status of such systems. The systems surveyed are: HP's BMS, Forest, ESF K/1's Software Bus, ESF Kernel/2r's Muse, Polylith, Arcadia's Q, Weaves. Other candidate systems will be added in later versions of this document.

A summary of the information in section 3 appears in table form in Appendix A.

2 SBUS Framework: Attributes/Characteristics

This section describes a reference framework, denoted SBUS framework, for the description and comparison of SEE software bus systems. It consists of a set of attributes or characteristics, described next. The objective of this framework is to help us understand the characteristics of such systems in order to define guidelines for using them in the construction of SEEs. As our investigations proceed, new attributes are being defined, and old ones are refined.

2.1 Primary Purpose

This attribute describes the primary purpose or objective of the software bus system (one or more software components).

2.2 Communication Model

This attribute describes the kind of communication model supported by the software bus, i.e., how do components communicate. There are many sub-characterizations of the ways components communicate, as described below.

1. *Direct vs. Mediated Communication.* This attribute describes whether the software bus provides a dedicated communication channel between a sender (of a message) and a receiver,

or the message is intercepted and delivered by an external agent.

2. *Point-to-point vs. Broadcast vs. Multi-cast.* This attribute typically applies to message-based systems. These elements are as follows:

   - Point-to-point. The sender explicitly identifies the receiver.
   - Broadcast. The message is broadcast to every component that may be listening in the environment
   - Multi-cast - There is some selection mechanism that identifies a subset of components to receive the message.

3. *Client/server or peer-to-peer.* This attribute describes whether, within a single thread of execution, the communication between two components is client/server or peer-to-peer.

   Client/server is an environment where the requester (client) of a service is on one system and the supplier (server) of the service is potentially on another system. Their arity may vary:

   - multi-client, where more than one client makes requests of the server;
   - multi-server, where one client makes multiple requests to multiple servers; and
   - multiple client and multiple server.

   Peer-to-peer is an environment where two clients (potentially on separate processors) can submit requests to one another over a single logical connection - that is, a single communications sequence over the network. Each client in this case can also be a server.

4. *Location Transparency.* This attribute describes whether a component needs to know the location of the component with which it communicates.

5. *Naming.* This attribute describes whether the components for communication need to be addressed explicitly. There are cases where abstract names can be used.

2.3 Run-time Behavior

This attribute describes the run-time characteristics supported.

2.4 Bus Interface Description

This attribute describes how components use or interface with the bus, e.g., via message-based, procedure-call-based semantics, or some variant. It also describes the interface and its parameters.

2.5 Binding Time

This attribute describes the binding time. "Early binding" implies static/compile time association whereas "late binding" implies run-time association. However, there can be several levels of binding. For example: i) When is the association between an abstract service name and the actual physical software that implements the service made? ii) When is the association between the service requester and the service provider made? iii) Does the service requester have to know apriori

what services exist in the environment? iv) Can services be dynamically added to an environment and be accessed by other components in a non-intrusive way (without recompilation or relinking of existing components)?

2.6 Data Granularity and Type

This attribute describes the granularity of the data being passed by the bus.

2.7 Process Granularity

This attribute describes the granularity of the process elements being communicated (e.g., routine, procedure, operating system process, Ada task).

2.8 Multilingual Support

This attribute describes whether the bus allows communication between components written in different languages, and, if so, what kind of language bindings are provided.

2.9 Data Translation

This attribute describes whether data translation is supported and how transparent this translation is to the communicating components.

2.10 Protocols

This attribute describes the underlying protocols used to implement the bus. It may describe whether the message content at different levels of abstraction conform with some standard protocol.

2.11 I/O Synchronization

A bus interface can be synchronous, asynchronous or both, as follows:

- Synchronous protocol - a communications protocol in which the component acting as client suspends execution of its current process until it receives a response from either the component acting as server or the software bus.
- Asynchronous protocol - a communications protocol in which the component acting as client does not wait for a response from the component acting as server even though one may be expected in due course (this implies that responses are handled either by polling or interrupts).

This attribute should also specify whether the requesting component blocks when communicating with other component via the bus.

2.12 Triggering

This attribute describes whether the software bus provides the capability to intercept messages on the bus and trigger actions based on those messages.

2.13 Threads

Threading refers to the ability to divide a program into multiple parts that execute concurrently within the same virtual address space. A multi-threaded program has multiple points of execution interleaving faster computational operations with slower operations. This attribute describes whether the bus is single or multi-threaded. Other applicable questions may deal with whether the server is re-entrant.

2.14 Scope

This attribute describes the scope of the bus, i.e., whether the bus is used for coordinating communication of components: a) within a single process family; b) across multiple process families within the same processor; or c) across multiple process families and across multiple processors.

2.15 Distribution

This attribute describes whether the bus supports distribution across multiple processors and whether this distribution is transparent to the components. It should also describe how distribution is specified and whether it is static or dynamic.

2.16 Component Interface Specification

This attribute describes whether there is linguistic support for specifying component interfaces. It should also describe whether it supports the generations of interface stubs.

2.17 Registration

This attribute describes how components are registered in the software bus environment. It should include information such as: a) whether the registration of components is static or dynamic; b) how components are changed or replaced; c) whether there is support for dynamic update of communicating components.

2.18 Exception Handling

This attribute describes how exceptions are generated, captured, handled, and/or propagated.

2.19 Security

This attribute describes the security model supported by the bus, if any. For example, whether there are access control mechanisms provided to ensure varying degrees of secure communication.

2.20 Versioning

This attribute describes any versioning support.

2.21 Software Bus Development Tools

This attribute describes the software bus tools (e.g., development, generative, analysis, browsing) which support the development of software bus applications.

2.22 Platform Dependencies

This attribute describes any hardware and/or software dependencies.

2.23 Strengths

This attribute describes the primary strengths of the architecture supported by this software bus.

2.24 Weaknesses

This attribute describes the main limitations of this architecture, e.g., hardware, software, language, error detection/correction, functionality, etc.

3 Systems Surveyed

This section presents a first draft of a survey of systems which have been characterized as software buses; it uses the framework described in Section 2. This survey is preliminary and based on a subset of the papers referenced. It is worth pointing out that sometimes different papers contradict each other since they reflect evolution of those systems but also, in some cases, the description may represent thought processes and concepts not yet implemented in the existing systems. More information about the SEEs which include software bus systems can be found in [KPS93].

Note that the attributes Exception Handling, Security and Versioning do not appear in the description of the systems, since they were added to the framework just before this document was published. Also note that some sub-sections were omitted intentionally; it indicates that either the information was not available or was not explicit in the documentation. For example, the attributes "Strengths" and "Weaknesses" do not have values since, at this stage in the investigation, we do not feel confident we really understand those systems.

3.1 BMS

The Broadcast Message Server (BMS) [BTJ89, Kra89, Cag90] was originally developed as part of the Hewlet Packard SoftBench environment but it works as an autonomous (set of) component(s) which have been ported to many platforms. Tools can communicate requests for action and can notify the completion of actions via the BMS. The broadcast nature of the BMS communication allows the set of tools managed by a BMS and interested in a particular message to be extended without requiring any change in the tools that send the messages.

Primary Purpose. The BMS is a broadcast message server which serves as a control integration framework for tools. Tool communication is based on selective broadcast.

Communication Model. The BMS supports the following communication model:

1. Mediated communication via the BMS.
2. Event-driven selective broadcast (single broadcast server per process family).
3. Multi-client, Multi-server, Multi-client/server.
4. Location transparent to communicating components.
5. Anonymous addressing (Sender is unaware of who the receivers are).

Run-time Behavior. Events occur in the system when tools send messages to the BMS Server. The BMS Server rebroadcasts these messages to all tools (or components) that have expressed an interest in that particular type of message.

Bus Interface Description. The BMS provides a message-passing interface with support for procedure call emulation. The interface parameters include: Request id, msg type (notification, request, or failure), command class, command name, context, and arguments.

Binding Time. It seems that the binding of name to component is done at tool registration time. However, any tool that registered an interest on a particular message is notified at run time.

Data Granularity and Type being transferred. The message arguments are character strings.

Process Granularity. The granularity is at the UNIX process or tool level.

Multilingual Support. No support. The language supported is C.

Data Translation. No provision. However, data translation can be accomplished by introducing translator components, activated using the same BMS mechanism.

Protocols. Implemented using Unix Sockets. An Abstract Tool Protocol is currently being standardized by CASE Communique to be used with message passing systems.

I/O Synchronization. It supports synchronous and asynchronous communication.

Triggering. The event-based mechanism can be viewed as a form of triggering capability. Users can define their own triggers with the Encapsulator.

Threads. Single threaded.

Scope. Scoping is identified by the triple (hostname, working_directory, filename).

Distribution. Yes, supported by the SPC (Software Subprocess Control). It is not clear whether BMS's running on multiple CPUs can communicate with each other.

Component Interface Specification. The Encapsulation Description Language (EDL) is used for encapsulating tools to run with the BMS. The BMS maintains tables that provide implicit association between components at run-time.

Registration. Apparently, tools are registered at compile time. However, it seems that information about which event a tool wishes to see or be notified are passed to the BMS when the tool is activated.

S/W Bus Development Tools. The Encapsulator [Dav89, Cag89a, Cag89b] provides a means of integrating tools into the HP Softbench user-interface and BMS. The encapsulation consists of the Encapsulation Description Language (EDL) which describes a user-interface and corresponding communication across the BMS. The communication information consists of messages which it will respond to and messages it will generate in response to user-interface events.

3.2 Forest

The Forest system [GI90] extends the Field and the HP BMS broadcast approach to support user defined control policies over tool interaction.

Primary Purpose. The basic idea is, instead of simply routing messages between tools, a message server consults an invocation policy description that determines how and when messages should pass between tools.

Communication Model. Its communication is similar to BMS's model with the addition of policy constrained broadcast, i.e.,

1. Mediated communication
2. Event-driven selective and policy constrained broadcast.
3. Multi-client, Multi-server, Multi-client/server.
4. Location transparent
5. Anonymous addressing

Run-time Behavior. Similar to BMS, but rather than simply routing messages between tools, the message server consults an invocation policy description that determines how and when messages pass between tools.

Bus Interface Description. It supports a message-passing interface. Its interface seems to be more generic than BMS'. The arguments are specified character string patterns.

Binding Time. It appears similar to the BMS system.

Data Granularity and Type being transferred. The message arguments are character strings

Process Granularity. The granularity is at the UNIX process or tool level.

Multilingual Support. No support. The language supported is C.

Data Translation. No provision.

Protocols. Implemented using Unix Sockets. Message pattern matching is based on condition-action pairs.

I/O Synchronization. It supports synchronous and asynchronous communication.

Triggering. It supports event-based and condition-based triggering.

Threads. Single threaded.

Scope. It seems to span multiple process families.

Distribution. Yes. Supported with SPC (Software Subprocess Control)

Component Interface Specification. It maintains tables that provide implicit association between components at run-time, together with condition-action pairs.

3.3 K/1 Software Bus.

The Eureka Software Factory (ESF) project[1] has a key objective to prepare the foundation for Software Factories in Europe. A major focus of the ESF project is on two critical enabling technologies: *process support*, the use of programmable models of factory activity to make the complex workflow of software development teams subject to accurate planning and continued fine-tuning; and *software componentry*, the use of abstract interfaces and environment standards (e.g., software bus) to enable "plug-in" access to multiple system platforms, thus encouraging more generic software design and more frequent software re-use. The "Software Bus" (SwB) [FNBG91, BFM, Fou90, ESF90] is an important concept of ESF environments, meaning an abstract communication channel which hides distribution and allows the exchange of data and control information among environment components.

Kernel/1 is considered to be the first industrial implementation of an ESF Factory Support Environment (FSE) Framework. It was built by three industrial partners: CAP Gemini Innovation, CAP debis GEI, and Sema Group. Kernel/1 consists of a set of software tools and libraries which

[1]ESF is a ten-year cooperative project, set up at the end of 1986 under the Eureka programme by a powerful European consortium of users, suppliers and research institutes.

support building, customizing, running and extending a software factory. It operates on networks of UNIX workstations.

The Kernel/1 Software Bus [Mor92, Mor93b, Mor93a, M+93] provides a channel through which interconnected components can communicate. It allows for components to be "plugged" in and out of the environment and to be replaced in a non-intrusive way. Components supply and/or use services of the Factory. At build time, the SwB provides means of describing the functionality of components in terms of abstract services they provide or require as well as their implementation details. The Kernel/1 SwB is an extension of the remote procedure call (RPC) mechanism which includes an abstract language ASN.1 and supports components written in C and Lisp; Ada and C++ are in the works. It was built by the Sema Group. It also provides facilities for notification of events, a CAP Gemini extension to the HP Softbench.

Primary Purpose. An industrial implementation of the ESF S/W Bus to provide an interaction mechanism between two or more Software Factory Components. It provides a channel through which interconnected components can communicate. It allows for components to be "plugged" in and out of the environment and to be replaced in a non-intrusive way.

Communication Model. Its communication model is as follows:

1. Direct communication, with the client identifying the server by an abstract service name.
2. Point-to-Point, where the client identifies specifically the server it wishes to connect to.
3. Client/Server. A component in K/1 can be a client or server.
4. Location transparent, where the client is not aware of where the server resides (local or remote), it only knows about abstract service names.
5. Explicit addressing by abstract service name.

Run-time Behavior. At run time, providers of services (servers) "register" with the Software Bus (via *SWB_Export()* calls). For each registered service, a *ticket* is created in the local service *Component Locator*'s ticket pool and will be made available for the first client component requesting the same service (via the *SWB_Import()* call). As soon as the server has successfully registered with the SwB, it goes into an infinite loop, listening for request on the service. When a client component registers with the *Locator* it receives the corresponding ticket which establishes the binding with the server component that exported the service. Until the client returns the ticket, each request issued by the client is guaranteed to be serviced by the same server.

Bus Interface Description. The SwB interface emulates the procedure call semantics. From the client program's perspective, a simple procedure call such as *SWB_Import(service_name)* allows a client to request a service. Control Exchange Statements (CES) are procedure call interfaces to the operations (methods) associated with the services. A client invokes an operation associated with a service via the CES (i.e., the programming language specific procedure call interface) defined for that operation.

Binding Time. Binding between an abstract service name and its implementation code is accomplished at compile and link time, therefore addition of new operations or modification of existing operations will require recompilation and relink.

Binding between the service requester (client) and the service provider (server) is dynamic. The *Locator* provides the run-time management of services and makes the association between a server exporting a service and the client requesting for that service through the assignment of tickets. Once the client holds the ticket to a server, all subsequent CESs are guaranteed connection to the same server.

Data Granularity and Type being transferred. Two categories of data types are supported: primitive (i.e., atomic) and constructed (i.e., structures whose elements are members of either primitive or constructed data types). The SwB provides a set of APIs for manipulating abstract data types. The abstract definition of these data types are expressed in the Service Abstract Description Languages (SADL). A set of pre-defined types are provided which include Array, Record, Choice, Set, Bag, Directed Graphs, Net, Tree, and List.

Process Granularity. The SwB supports both procedure and Unix process level granularity. An abstract service typically provide multiple operations that are defined in the Service Abstract Description Module in SADL. Associated with each operation there is a corresponding procedure that implements the semantics of the operation. The procedure can in turn invoke a separate executable if so desired.

Multilingual Support. It supports multiple languages, C, C++, Le-LISP.

Data Translation. The SwB implements the concept of "plugs" which perform the encoding and decoding of arguments before and after transmission. There are two layers of software that constitute the communications interface part of the Kernel/1 SwB. The first layer, called the *Plug*, is the language dependent portion that describes the data type and the CES of a service in terms of the programming language representation of its component. It handles data conversion (from client internal representation to external representation, and from external representation to server internal representation). A *Plug* is specific to a given installed Component. The second layer, called the *Agent*, is the underlying language independent software that marshals/unmarshals data to/from the form of transmission on the SwB communication channel. The Agent also calls the procedure which corresponds to the Request received from the client.

Protocols. Kernel/1 Components communicate with the *Locator* through Unix named pipes. *Locators* communicate between themselves across the network through UDP Sockets. *Agents* are part of the communication interface implemented on top of XDR/RPC.

I/O Synchronization. The Kernel/1 Software Bus interface can be both synchronous and asynchronous. It is not clear from the documentation whether the mode of synchronization can be controlled by the client program, or how and where the synchronization mode is specified.

Triggering. It does not support triggering.

Threads. Multi-threading is an optional feature via SunOS Light Weight Processes (LWP). Primitives are provided to attach a service instance to an LWP thread; different threads can therefore offer different service instances without having to handle, in one thread, events concerning another thread.

Scope. There is one Software Bus *Locator* executing and one run-time environment for each processor. The service name space therefore is assumed to be flat; all service names must be unique within a distributed environment. It is not clear from the documentation whether there can be multiple implementations of the same service. For example, there might be two different editors (e.g., *vi* and *Emacs*) available in the environment that can potentially offer the same *Editing* service. It is not clear whether and how a user can dynamically switch between *vi* and *Emacs*. It also is not clear how multiple service instances can execute currently if a ticket associated with a Server must be returned before the next instance can initiate.

Distribution. The location of a service is transparent to the client application. However, it is not clear from the documentation where and when the location of a Server is specified; whether it is at compile time or at Component installation time; or whether the Server location can be changed at run-time.

Component Interface Specification. Kernel/1 Software Bus provides three separate languages for describing Software Bus Components:

- Component Type Description Language (CTDL) - a textual description of a Component Type which specifies those services this Component exports and imports.
- Service Abstract Description Language (SADL) - a textual specification of the machine independent aspects of an FSE service. It includes an abstract description of the interface to its operations, a semantic description of the operations, and a description of its Universe of Discourse (a set of concepts that are known to a given component.)
- Service Representation Description Language (SRDL) - a formal language in which the implementation description of a service and its data types according to the Component role (client or server) and programming language. It furnishes the mappings between the abstract operations and the actual procedure or function calls found in the Component application code. There is one SRD module for each language implementation of a service.

Registration. It seems to support dynamic addition of new components but not change of existing components.

One of the first things that an executing Component must do, is to register its presence with the Software Bus run-time system. This is accomplished through the *SWB_Export* call. The *Locator* verifies that the Component Type is known in its Component Type database. If the verification process is successful, then the server Component is provided with a ticket which it will use when binding with the client Component requesting its services.

S/W Bus Development Tools. The Kernel/1 SwB development facility include the compilers for CTDL, SADL, and SRDL; and the Software Bus library routines.

Platform Dependencies. The Kernel/1 SwB runs on the SPARC, Sun3, Sun4, and PC (under MS Windows/3.1)

3.4 K/2r MUSE Software Bus.

Kernel/2r [AD+92, AH, Uni91] or K/2r is a research prototype of an ESF Factory Support Environment; it was developed at the University of Dortmund, Germany. It supports a layered distributed computing architecture designed to disperse services and data across an enterprise network. MUSE [Hol92a, Hol92b, Sch91, HS91] is K/2r's software bus. It is the nucleus of the K/2r environment that connects the process components and service components. It also implements a number of message handling systems, not just one solution like HP's Broadcast Message Server.

Primary Purpose. MUSE is a research prototype implementation of the ESF S/W Bus, whose primary purpose is to support architectural extensibility through the concept of "plug and play" that enables the plug-in and interplay of new architectural components with the software bus. Its main purpose is to manage the interoperation of components in a distributed environment by means of autonomous transaction processing.

A MUSE instance basically consists of an S-transaction repository, an S-transaction Handler (the S-transaction Definition Language (STDL) interpreter), an application interface, and an interface to the encapsulated service components. S-transactions are provided in order to describe the cooperation of autonomous components, located at geographically dispersed sites. For more on S-transactions, see [VEH91].

Communication Model. MUSE's communication model supports the concept of transaction-based communication; it is as follows:

1. Server mediated communication.
2. Point to point.
3. Peer-to-peer.
4. Location transparent.
5. Explicit addressing by abstract service name.

Run-time Behavior. An execution plan, defined using the S-transaction definition language (STDL), is defined at tool installation time and stored in the S-transaction repository at all participating sites. An S-transaction interpreter/handler takes these plans and processes the S-transactions by initiating the required local and remote transactions.

Bus Interface Description. An Interoperation Description Language (IDL), based on the S-transaction Definition Language (STDL) [MH91, VEH91] is provided as the vehicle for specifying cooperation between autonomous components in a distributed environment. The underlying S-transaction system model is that of a set of cooperating peer components where the relationship between the components are defined with S-transaction type definitions. An S-transaction type definition contains an execution plan specifying what requests are to be processed where in the network. This type definition is expressed in STDL.

Binding Time. The inter-relationships between components are dynamically defined within S-transaction type definitions. The type definitions, encoded in STDL scripts, are defined at installation time and stored in a repository at all participating sites. These type definitions contain an execution plan indicating what requests are to be processed where (i.e., local or remote).

Data Granularity and Type being transferred. It supports C data types.

Data Translation. Data translation can be defined at registration time and it is transparent to applications.

Protocols. The MCS (Multi-Communications System) on top of X400, TCP/IP, RPC.

I/O Synchronization. It supports synchronous communication.

Triggering. It seems that triggering can be programmed as S-transactions.

Threads. It is single threaded.

Scope. MUSE spans multiple process families.

Distribution. Yes, and distribution is transparent to applications.

Component Interface Specification. Components use the STDL language.

Platform Dependencies. SUN4 under UNIX.

3.5 Polylith Software Bus.

The Polylith system [Pur90, PSW91, CWP92, CH90] is a software interconnection system. It allows programmers to configure applications from mixed-language software components and then execute those applications in diverse environments.

Programmers specify components in terms of a Module Interconnection Language (MIL); Polylith uses this specification to guide packaging (static interfacing activities such as stub generation, source program adaptation, compilation and linking). At run time, an implementation of the bus abstraction may assist in message delivery, name service or system reconfiguration.

Primary Purpose. Components can be implemented separately from the implementation of its interfaces. Interfacing decisions can be encapsulated separately, using a software bus.

Communication Model. Polylith's communication model is as follows:

1. Direct Communication with optional mediation.
2. Multicast.
3. Multi-server, Multi-client, Multi-client/server.
4. Location explicitly specified in a Module Interconnection Language (MIL) but not in an application interface.
5. Addressing explicitly specified in MIL but not in application interface.

Bus Interface Description. Polylith supports primarily a message passing paradigm but it also supports a procedure call interface.

Binding Time. It supports early binding, statically specified in MIL. However, it seems that later versions support dynamic binding.

Data Granularity and Type being transferred. It supports: a) primitive types: integer, string, boolean, and float; b) composite types that are based on combination of primitive types using vectors and record structures; c) Capability type; and d) Raw type: uninterpretable by any interconnection substrate.

Multilingual Support. It supports multiple languages, C, C++, Ada, Pascal, Lisp, Prolog.

Data Translation. It seems that data translation can be defined via the MIL and it is transparent to applications.

I/O Synchronization. It support synchronous and asynchronous communication.

Triggering. No support for triggering.

Threads. Single threaded.

Scope. Flat space.

Distribution. It supports distribution.

Component Interface Specification. It provides a Module Interconnection Language for interface specification.

3.6 Q

The Q system [Hei92, MH+92, May92] is an important component in the Arcadia set of environment components. It is an enhanced remote-procedure-call component which allows a client to invoke an arbitrary server dynamically.

The Arcadia Research Project is being conducted by the Arcadia Consortium to develop innovative technology in support of advanced SEEs. It is important to observe that there is not one Arcadia environment but a combination of technology and components in support of environment building.

Primary Purpose. The objective of the Q system is to support the interconnection of multilingual software components for execution in heterogeneous environments.

Communication Model. The Q system supports the following communication model:

1. Direct communication
2. Point-to-point communication
3. Client/server system, supporting multi-client, multi-server, or multi-client/server.
4. Location. Client is aware of the location of the server.
5. Naming. Explicit addressing.

Run-time Behavior. The client issues a service request by sending a message to the server process. The client can continue processing, if appropriate. The server remains inactive until it is awakened through a signal based notification mechanism to process the request.

Bus Interface Description. Q supports a procedure call interface. The client specifies: server (machine, server id, version), service type, and Q-data representation (QDR) buffer.

Binding Time. Q supports late binding.

Data Granularity and Type being transferred. Q supports both primitive types (i.e., integer, string, boolean, and float) and composite types that are based on combination of primitive types using vectors and record structures.

Process Granularity. Q supports communication between UNIX processes.

Multilingual Support. Q supports multiple languages, C, Ada, C++, E, LISP, and Prolog.

Data Translation. In Q, data translation is not transparent to the application. The Q client explicitly encodes its arguments prior to issuing a remote procedure call (RPC) and explicitly decodes any returned results.

Protocols. Q is implemented on top of modified RPC (ARPC). It also depends on Sockets, TCP/IP, UDP. It uses a data representation protocol (QDR).

I/O Synchronization. Q supports synchronous and signal-based asynchronous communication. In signal-based asynchronous communication, the sockets are configured for asynchronous I/O. When a service request arrives at the socket, a signal is sent to the service dispatcher. The service dispatcher will call the appropriate service procedure to furnish the service requested. The service procedure may "acknowledge" completion of the service allowing the client to continue execution concurrently with the service procedure.

Triggering. There is no support for triggering.

Threads. Q is single threaded for C and multi-threaded for Ada. It was designed to support concurrently active communicating servers.

Distribution. Q supports distribution but it is not transparent to application.

S/W Bus Development Tools. QGen is a Q stub generator.

Platform Dependencies. SUN XDR/RPC.

3.7 Weaves

Weaves [GR91] are networks of concurrently executing tool fragments that communicate by passing objects. Its architectural characteristics lie between heavyweight parallel processes and fine-grain dataflow.

Tool Fragments are small software components on the order of a procedure that perform a single well-defined function. Each fragment executes as an independent thread, a light weight concurrent process that shares memory with others of its kind.

Primary Purpose. The Weaves system is intended as an engineering medium for systems characterized by streams of data. Its emphasis is on instrumentation, continuous observability and dynamic rearrangement.

Weaves differs from Unix pipes in that: Weaves tool granularity is finer (i.e., at the procedure level); Weaves process granularity is finer (i.e., light weight processes); Weaves permit multiway communication; and Weave streams are structured.

Communication Model. Weaves' communication model is as follows:

1. Indirect communication via shared data (i.e., "queues").
2. Medium-grain, non-blocking data flow with finite buffering, FIFO stream access, and multiple readers and writers.
3. Peer-to-peer (does not care where the message came from or where it goes).

4. Location transparent.

5. Communication is blind, i.e., no tool in a weave can tell where its input objects come from or where its output objects go; no tool knows what form of transport service is being used; and no tool is aware of a loss of connection.

Run-time Behavior. Weaves are a network of concurrently executing tool fragments that communicate by passing objects. Objects are transmitted from one tool fragment to another via ports attached to queues. The queues, in turn, buffer and synchronize communication among tool fragments.

Objects are passive flowing through the weaves. Only tool fragments are active, accepting objects from ports, invoking the methods implemented by the objects, performing tool-specific computations and passing objects downstream.

Bus Interface Description. Weaves supports a message-passing interface. Weaves is implemented as a C++ object library.

Binding Time. Weaves supports continuous incremental change; therefore, it supports late or dynamic binding. Weaves are extended without disturbing the behavior of the tool fragments in place.

Data Granularity and Type being transferred. Weaves support medium grain data. Each individual stream datum is an object with encapsulated state, class membership, inheritance and exported methods.

Process Granularity. It supports procedure level granularity.

Multilingual Support. It supports C++ and other languages which interface with C++.

Data Translation. In Weaves, ports implement the wrapping and unwrapping of data objects by means of envelopes which hide the type of underlying data object. A specialized port can be used to mediate communication between multi-lingual tool fragments.

I/O Synchronization. It supports asynchronous communication.

Triggering. It does not support triggering.

Threads. Multi-threaded; implemented as C++ layering over Sun Light Weight Process Library.

Distribution. It does not support distribution.

Component Interface Specification. It does not provide a language for interface specification.

Registration. It supports dynamic weaving; the weaves can be dynamically rearranged without disturbing the flow of the objects. New tool fragments can be added without concern for detail of interconnection or integration.

Platform Dependencies. Sun3, InterViews, Xwindows.

Major Weaknesses. A missing capability is an Interface Description Language which hides the low-level aspects of communication and describes the imported and exported functions and data types of the real services and clients.

4 Conclusions.

In this document we presented the SBUS framework, a framework for the understanding and comparison of systems considered as SEE software buses; it includes an initial set of attributes and characteristics. We have also performed an incomplete survey of a few of such systems and characterized them based on the available documentation. (It is worth noting, though, that we have listed a superset of the papers used in this document, since some of these papers were received recently and therefore not incorporated).

This document is just the beginning of an investigation towards understanding the complex issue of componentization and interconnectivity of heterogeneous components. Much more work lies ahead of us before we can fully understand when and how those components should be interconnected. Other candidate systems mentioned in literature but not addressed here are: DCE, Ole2, SCORPION, CORBA, ToolTalk, Matchmaker/MIG, Mercury, Isis, SLI, Abe, Conic, Durra, Infuse, HPC/HRPC (Heterogeneous remote procedure call), Mercury, MLP (mixed language programming). We plan to investigate those systems and characterize them using the SBUS framework.

5 Acknowledgement.

This work was supported by the Advanced Research Projects Agency, ARPA Order No. 7314, issued by the Space and Naval Warfare Systems Command under contract #N00039-91-C-0151.

References

[AD+92] R. Adomeit, W. Deiters, et al. K/2R: A Kernel for the ESF Software Factory Support Environment. In *2nd International Conference on Systems Integration 92*, Morristown, NJ, June 1992.

[AH] R. Adomeit and B. Holtkamp. ESF Factory Support Environment: Architectural Refinements and Alternatives. Technical report, University of Dortmund.

[BFM] T. Bingen, R. Foulkes, and L. Morgan. Data and Control Integration in a Software Factory: in Software Bus. Technical report, Sema Group - Brussels, Glasgow, Paris.

[BTJ89] Jr. B. T. Jenings. The HP SoftBench Message Model: Concepts and Conventions Used by the HP SoftBench Tools. SoftBench Technical Note Series SESD-89-21 Revision 1.2, Hewlett-Packard, Software Engineering Systems Division, 3404 E. Harmony Road, Fort Collins, Colorado 80525, September 1989.

[Cag89a] M. Cagan. An Encapsulator Tutorial. SoftBench Technical Note Series SESD-89-15 Revision 1.1, Hewlett-Packard, Software Engineering Systems Division, 3404 E. Harmony Road, Fort Collins, Colorado 80525, August 1989.

[Cag89b] M. Cagan. The McCabe Encapsulation. SoftBench Technical Note Series SESD-89-18 Revision 1.0, Hewlett-Packard, Software Engineering Systems Division, 3404 E. Harmony Road, Fort Collins, Colorado 80525, August 1989.

[Cag90] M. Cagan. The HP SoftBench Environment: An Architecture for a New Generation of Software Tools. *Hewlett-Packard Journal*, June 1990.

[CH90] Purtilo C. Hofmeister, J. Atlee. *Writing Distributed Programs in Polylith.* University of Maryland, November 1990.

[CWP92] C. Chen, E. L. White, and J. M. Purtilo. A Packager for Multicast Software in Distributed Systems. Technical report, University of Maryland, 1992.

[Dav89] H. Davidson. Encapsulator: The Plug-In Compatibility Tool for SoftBench. SoftBench Technical Note Series SESD-89-11 Revision 1.1, Hewlett-Packard, Software Engineering Systems Division, 3404 E. Harmony Road, Fort Collins, Colorado 80525, June 1989.

[ESF90] Software Bus - Rationale. Technical report, ESF - SwB Sub-Project, June 1990.

[FNBG91] R. Foulkes, O. Nanlot, T. Bingen, and C. Ginn. The Software Bus - An Overview. Technical report, Sema Group Belgium, September 5, 1991 1991.

[Fou90] R. Foulkes. The ESF Software Bus. Technical report, Yard Ltd., UK - Glasgow, 1990.

[GI90] D. Garlan and E. Ilias. Low-cost, Adaptable Tool Integration Policies, for Integrated Environments. In *4th ACM SIGSOFT Symposium on Software Development Environments*, Irvine, CA, December 1990.

[GR91] M. Gorlick and R. Razouk. Using Weaves for Software Construction and Analysis. In *13th International Conference on Software Engineering*, Austin, Texas, May 1991.

[Hei92] D. H. Heimbigner. ARPC: An Augmented Remote Procedure Call System. Technical Report CU-Arcadia-100-92, Department of Computer Science, University of Colorado, October 19 1992.

[Hol92a] B. Holtkamp. MUSE - A Framework for Decentralized Software Development Environments. Technical report, University of Dortmund, 1992.

[Hol92b] B. Holtkamp. MUSE Interoperation Support. Technical report, Fraunhofer Institue for Software and Systems Technology, 1992.

[HS91] B. Holtkamp and F. Schuelke. The Software Bus - Communication Aspects. Technical report, University of Dortmund, March 18 1991.

[KPS93] A. Karrer, M. H. Penedo, and C. Shu. A Survey of Software Engineering Environment Architecture Approaches. Technical Report Arcadia-TRW-93-007, TRW, Redondo Beach, CA, 1990, November 1993.

[Kra89] S. A. Kramer. SoftBench DM Message Integration Requirements. SoftBench Technical Note Series SESD-89-20 Revision 1.2, Hewlett-Packard, Software Engineering Systems Division, 3404 E. Harmony Road, Fort Collins, Colorado 80525, August 1989.

[M+93] L. Morgan et al. *The Software Bus - SADL Reference Manual.* Sema Group Belgium, BAeSema, swb/doc/sadiref/version1.3 edition, January 1993.

[May92] M. J. Maybee. *Q: A Multi-lingual Interprocess Communications System - Reference Manual*, February 1992.

[MH91] B. Holtkamp Mm. Hallmann. STDL: A Definition Language for Semantic Transactions. Technical Report NIST Special Publication 500-201 and ECMA/TC33 Technical Report TR/55, 2nd Edition, NIST, December 1991.

[MH+92] M. J. Maybee, D. H. Heimbigner, et al. Q: A Multi-lingual Interprocess Communications System for Software environment Implementation. Technical report, Department of Computer Science, University of Colorado, 1992.

[Mor92] L. Morgan. *The Software Bus - User Requirements.* Sema Group Belgium, 4.2 edition, September 1992.

[Mor93a] L. Morgan. *The Software Bus - Reference Manual.* Sema Group Belgium, 4.0 edition, March 1993.

[Mor93b] L. Morgan. *The Software Bus - User's Guide.* Sema Group Belgium, 4.0 edition, March 1993.

[PSW91] J. M. Purtilo, R. T. Snodgrass, and A. L. Wolf. Software Bus Organization: Reference Model and Comparison of Two Existing Systems. Technical Report Technical Note No. 8, DARPA Module Interconnection Formalism Working Group, November 1991.

[Pur90] J. M. Purtilo. The Polylith Software Bus. Technical Report 2469, University of Maryland, 1990.

[Sch91] F. Schuelke. The MUSE System. Technical report, Eureka Software Factory, April 25 1991.

[Uni91] UniDo. Kernel/2 Definition - Draft Version. Technical report, ESF, 1991.

[VEH91] J. Veijalainen, F. Eliassen, and B. Holtkamp. *The S-transaction Model.* Morgan Kaufmann, July 12 1991.

A Software Bus Survey Highlights.

See attached

Software Bus Survey Highlights

	BMS	Forest	K/1 Software Bus	K/2r MUSE Software Bus	Polylith Software Bus	Q	Weaves
Primary Purpose	Control integration framework. Tool integration based on selective broadcast.	Control integration framework. Extended the BMS integration scheme to support user defined control policies over tool interaction.	An industrial implementation of the ESF S/W Bus for K/1: To provide an interaction mechanism between 2 or more Software Factory Components.	A research prototype implementation of the ESF S/W Bus for K/2r : To manage the interoperation of components in a distributed environment by means of autonomous transactions processing.	A software interconnection system that supports the interconnection of diverse (mixed-language) software components for execution in heterogeneous environments.	A software interconnection system that supports the interconnection of diverse (mixed-language) software components for execution in heterogeneous environments.	Weaves are a network of concurrently executing Tool Fragments that communicate by passing Objects. Emphasis is on instrumentation, continuous observability and dynamic rearrangement.
Communication Model	Mediated Communication via the BMS. Event-driven selective broadcast. Single Broadcast Server per process family. Multi-client, multi-server, multi-client/server. Location transparent Anonymous addressing. Sender is unaware of the location and number of receivers.	Mediated Communication via the BMS. Event-driven selective broadcast. Single Broadcast Server per process family. Multi-client, multi-server, multi-client/ server. Location transparent Anonymous addressing. Sender is unaware of the location and number of receivers.	Direct Communication Point-to-point with the client identifying the specifically the server it wishes to connect to. Client (UIC)/Server (SC). A Component in K/1 can be a client or server. Location transparent. Explicit addressing by abstract service name.	Server mediated communication. Point-to-point. Peer-to-peer. Location Transparent. Explicit addressing	Direct communication with optional mediation. Multicast Multi-server, multi-client, multi-client/server. Location explicitly specified in Module Interconnection Language (MIL) but not in application interface. Explicit addressing specified in MIL but not in application interface.	Direct communication. Point-to-point communication. Multi-client, multi-server, or multi-client/server. Client is aware of location of server. Explicit addressing.	Indirect communication via shared data (through the use of queues). Medium-grain, non-blocking data flow with finite buffering, FIFO stream access. and multiple readers and writers. Peer-to-peer. Location transparent. Communication is "blind" - no tool in a weave can tell where its input objects come from or where its output objects go.
Run-time Behavior	Events occur in the system when tools send messages to the BMS Server. The BMS Server rebroadcasts these messages to all tools that have expressed an interest in each type of signal.	Similar to BMS, but rather than simply routing massages between tools, the message server consults an invocation policy description that determines how and when messages pass between tools.	A server registers its presence with the SwB with SWB_Export() call. This creates a ticket in the local "Locator's" ticket pool and makes it available for the 1st client component importing the same service.	The S-transaction interpreter/handler is the execution engine of MUSE. Execution plans, specified in S-transaction definition language (STDL) is defined at tool installation time and stored in the S-transaction repository. It contains the necessary information for the handler to initiate local and remote transactions as service requests come in.		Client issues a service request by sending a message to the server process. The client can continue processing if appropriate. The server remains inactive until it is awakened through a signal based notification mechanism to process the request.	Weaves are a network of concurrently executing Tool Fragments that communicate by passing Objects. Objects are transmitted from one tool fragment to another via ports attached to queues. The queues, in turn, buffer and synchronize communication among tool fragments.

Bus Interface Description	Message-based interface includes: Request id, msg type (notification, request, or failure), command class, command name, context, and arguments.	Message-based interface. Interface is more generic message interface than BMS. Arguments are specified character string patterns.	Procedure-call interface. Control Exchange Statements and Software Bus Interface routines: swb_Import(); swb_Export(), etc.	An Interoperation Description Language (IDL) is provided as the vehicle for specifying cooperation between components.	Primarily message passing paradigm but supports also procedure call interface.	Procedure call interface. Client specifies: server (machine, server id, version), service type, and QDR buffer.	Weaves is implemented as a C++ object library.
Binding Time	Binding of name to component appears to occur at registration time.	Similar to BMS	Binding betw. an abstract service name and its implementation is accomplished at compile and link time.	Inter-relationships between components are dynamically defined within S-transaction type definitions.	Binding statically specified in MIL. Later versions might differ.	Supports run-time binding.	Supports continuous incremental change through dynamic binding.
Data Granularity and Type Being Transferred	Message arguments - Character strings	Message arguments - Character strings	Two categories of data types: primitive (i.e., atomic) and constructed (i.e. structures whose elements are members of either primitive or constructed data types).	C data types.	Primitive types: integer, string, boolean, and float; Composite types: that are based on combination of primitive types using vectors and record structures; Capability type; Raw type: uninterpretable by any interconnection substrate.	Primitive types: integer, string, boolean, and float; Composite types: that are based on combination of primitive types using vectors and record structures; in C, C++, E, LISP and Ada	"Medium grain, data-flow architecture." Each individual streams datum is an object with encapsulated state, class membership, inheritance and exported methods.
Process Granularity	At Unix process or tool level	At Unix process or tool level	At procedure and process levels.			At Unix process level.	At procedure and process levels.
Multilingual Support	No. Only supports C	No. Only supports C	Yes. Supports C, C++, Le-LISP.		Yes. Supports C, C++, Ada, Pascal, Lisp, Prolog.	Yes. Supports C, Ada, C++, E, LISP, and Prolog	Yes. Supports C++ and other languages that interface with C++.
Data Translation	No direct support.	No support.	Client and server "Plugs" performs the encoding/decoding of arguments before and after transmission.	Transparent to application.	It seems that data translation can be defined via the MIL thus can be transparent to applications.	Not transparent to the application. The Q client explicitly encodes its args prior to issuing an RPC and explicitly decodes any returned results.	In Weaves, ports implement the wrapping and unwrapping of data objects by means of envelops which hide the type of underlying data object. A specialized port can be used to mediate comm. between multi-lingual tool fragments.
Protocols	Implemented using Unix Sockets. Abstract Tool Protocol - being standardized by CASE Communique.	Implemented using Unix Sockets. Message pattern matching based on condition-action pairs.	Components talk to Locators through named pipes. Locators communicate between themselves through UDP Sockets. Agents are part of the communication interface implemented on top of XDR/RPC.	MCS (Multi-Communications System) on top of X.400, TCP/IP, RPC, etc.		Q implemented on top of modified RPC (ARPC) and XDR (QDR). Also depends on Sockets, TCP/IP, UDP.	
I/O Synchroni-zation	Synchronous and Asynchronous	Synchronous and Asynchronous	Synchronous and Asynchronous.	Synchronous	Synchronous and Asynchronous.	Synchronous and Signal-based asynchronous.	Asynchronous
Triggering	Users can define their own triggers with the Encapsulator	Supports event-based and condition-based triggering.	Not supported.	Triggering might be programmed as S-transactions.	Not supported.	Not supported.	Not supported.

Threads	Single threaded.	Single threaded.	Multi-threading is an optional feature via SunOS Light Weight Processes (LWP).	Single threaded	Single threaded.	Single threaded for C. Multi-threaded for Ada. Designed to support concurrently active communicating servers.	Multi-threaded; implemented as C++ layering over Sun Light Weight Process Library.
Scope	Scope identified by triple (hostname, workding_directory, filename)	Spans multiple users	Spans multiple users.	Multiple users.	Flat space.		
Distribution	Yes. Supported with SPC (Software Subprocess Control)		Yes. Location of service is transparent to an application.	Yes. Transparent to application.	Yes.	Yes, but not transparent to application.	No.
Component Interface Specification	User of EDL to encapsulate tools. BMS maintained tables that provide implicit association between components at run-time.	Maintains tables that provide implicit association between components at run-time	Yes. CTDL (Component Type Description Language), SADL (Service Abstract Description Language), SRDL (Service Representation Description Language).	STDL	Module Interconnection Language		No.
Registration	Apparently tools are registered at compile time.		Supports dynamic addition of new components but not change of existing components.				Supports dynamic weaving; the weaves can be dynamically re-arranged w/o disturbing the flow of the objects. New tool fragments can be added w/o concern for detail of interconnection or integration.
Exception Handling							
Security							
Versioning							
Software Bus Development Tools	Provides Encapsulation Description Language and library routines for communication with BMS server.		(S)RDL, SADL, CTDL compilers.			QGen (Q stub generator)	
Platform Dependencies			SPARC, Sun3, Sun4 and PC under MS-Windows/3.1			Sun XDR/RPC.	Sun3, InterViews, Xwindows.
Major Strength							
Major Weaknesses							Missing is an IDL which hides the low-level aspects of communication and describes the imported & exported functions and data types of the real services and clients.

Draft Interoperability Working Group Comparison Matrix
D. Heimbigner
December 1994

==========
Definition of Matrix Columns (Characteristics):
==========

Administrative Information

Provider(s):
Contact Information:
Availability:
Cost:
Platforms (Hdw &/or OS):
Software dependencies:
Version:
Source of Comparison:

Technical Features

Component Coupling Model:
What model of distributed computation is supported?
Two kinds are currently distinguished:

Autonomous: services are defined and instantiated independent of any particular client. Note that this implies that binding of client to server is late binding.

Tightly integrated: services and clients are defined as components within a larger distributed computation.

Communication paradigms:
What styles of communication are supported between clients and servers?

RPC - remote procedure call, possibly with futures, in which a reply is expected for each request.

Messages - messages are sent from client to server, but with no necessary expectation of a matching response.

Broadcast - messages are broadcast with no necessary destination server.

Standards Adherence:
To what standards does this system adhere?

Multi-Protocol:
Can this system communicate using more than one protocol? Note that this is not intended to refer to the lowest level protocols such as TCP/IP or UDP. It is assumed that all of these systems use (at least) TPC/IP.

Interface Compiler:
Is there a compiler to take an interface specification and generate client stubs and/or server skeletons.

Languages Supported:
In which languages can clients and servers be written? Note that this does not imply general multi-language support.

Multi-Language Support:
Can clients and/or servers written in arbitrary languages? Additionally, can clients and servers written in different

languages be intermixed?

Dynamic Interfaces:
Is it possible for a client to construct and invoke a request dynamically? Is it possible for a server to receive a request for which it does not have an explicit interface method defined?

Location Transparency:
Is the location of the machine on which an object resides hidden from a client?

Object granularity:
Can an object transparently be both local within an address space as well running as a server in a separate address space? The alternative is to require clients to be aware that an object is non-local.

First Class Contained objects:
For objects contained within other objects, is it possible to access the contained objects using the same mechanisms as for non-contained objects?

Orthogonality:
Is it possible to servers and clients to address spaces (i.e. Operating system processes) in arbitrary combinations?

Combined Client+Server:
Can a server also be a client to other servers?

Fault-Tolerance:
To what degree can a client and/or a server recover from failures by other components? The set of possibilities are: (1) loss of connection, (2) non-transparent server replacement, (3) transparent server replacement. Obviously this set does not do justice to the extensive capabilities of a system such as ISIS, which has focused specifically on this issue.

Exceptions:
Does the communication protocol allow for propagating some form of exception from server back to client as part of a response to a request?

Authorization Control:
Is it possible to provide authentication information with requests?

Request Priorities:
Is it possible to prioritize requests to a server, as opposed to strictly FCFS?

Receipt Acknowledgement:
Is it possible for a server to tell a client that its request was received and then later provide the actual reply? Note that this is usually only useful for unreliable protocols such as UDP.

Futures:
Does the client side require use of the synchronous procedure call abstraction, or can a client use some form of futures mechanism to continue after starting a call and later receive the reply?

Threadable:

Is it possible to use this system in a multi-threaded executable?

Non-blocking substrate:
Is support for threading separable from the functionality of client-side procedure invocation and also from the functionality of server-side method invocation? This is necessary in order to embed the system into languages with non-standard threading mechanisms (e.g., Ada).

Separate Marshaling:
Is it possible to marshal arguments to a remote procedure separately from the process of invoking the procedure? This feature is necessary to support languages without callback.

Versioning:
Is it possible to support versions of the same object or interface?

Registration:
Is registration of an object/interface done statically (i.e., by a separate tool) or dynamically at runtime by the object itself?

Notes:

---

1. Note that I am separating out two related groups of systems: 1) The tightly integrated systems such as Polylith, PVM, and Mentat, and 2) the broadcast message bus systems such as ToolTalk, Field, and FUSE. In both cases, this segregation is deliberate since I believe that they should be the topic of separate discussions. Especially note that I recognize that there are distinctions within such groups. Thus, ToolTalk and Field have many characteristics which differentiate them from each other. Also, for example, Polylith and PVM support different binding models. Never the less, for the purposes of this matrix, I treat them as part of a single aggregate group. The matrix includes a line for each group and is there as a placeholder for the group.

2. This matrix is intentially not exhaustive. All of the systems described here, except for the aggregate entries, may share many features not listed in the matrix. The entries in the matrix were chosen mostly to illustrate differentiators between the systems, or to hi-light features thought to be most important (even if they were shared by all systems). The interface compiler is an example of the latter. All of the listed systems have some form of interface compiler, but it is considered such an essential feature that it is included explicitly in the matrix. The supported type systems for remote procedure arguments are mostly identical, and so are an example of a common assumption currently left out of the matrix.

SECTION I: Research Systems

Q

Provider(s): University of Colorado
Contact Information:
http://www.cs.colorado.edu/homes/arcadia/public_html/q.html
Availability: Public
Cost: Free
Platforms (Hdw &/or OS):
Solaris1, Solaris2, AIX, ULTRIX, OSF/1, HPUX, IRIX
Software dependencies: ARPC.
Version: 3.2
Source of Comparison: Local Expertise

Component Coupling Model: Autonomous
Communication paradigms: RPC
Standards Adherence: SUN ONC
Multi-Protocol: No
Interface Compiler: Yes (C and Ada)
Languages Supported: C, C++, Ada, Prolog (obsolete), Lisp (Obsolete)
Multi-Language Support: Yes
Dynamic Interfaces: Client-side, Server-side
Location Transparency: Yes
Object Granularity: Non-local
First Class Contained objects: No
Orthogonality: Yes
Combined Client+Server: Yes
Fault-Tolerance: Connection Loss Detection
Exceptions: Yes
Authorization Control: Yes
Request Priorities: Yes
Receipt Acknowledgement: Yes
Futures: Yes
Threadable: Yes
Non-blocking substrate: Yes
Separate Marshaling: Yes
Versioning: Yes
Registration: Dynamic

ILU

Provider(s): Xerox Parc
Contact Information:
ftp://parcftp.parc.xerox.com/pub/ilu/misc/janssen.html
Availability: Public
Cost: Free
Platforms (Hdw &/or OS): Solaris1, Solaris2, IRIX
Software dependencies: None
Version: 1.6.4
Source of Comparison: 1.6.4 Reference Manual

Component Coupling Model: Autonomous
Communication paradigms: RPC
Standards Adherence: SUN ONC, Courier, CORBA 1.2 (Partial)
Multi-Protocol: Yes
Interface Compiler: Yes
Languages Supported: C, C++, Modula-3, CLisp

Multi-Language Support: No
Dynamic Interfaces: No
Location Transparency: Protocol dependent
Object Granularity: Non-local
First Class Contained objects: No
Orthogonality: Unknown
Combined Client+Server: Yes
Fault-Tolerance: Connection Loss Detection
Exceptions: Yes
Authorization Control: Yes
Request Priorities: No
Receipt Acknowledgement: No
Futures: Yes
Threadable: Yes
Non-blocking substrate: No
Separate Marshaling: No
Versioning: Yes
Registration: Dynamic

==========
SECTION II: CORBA 1.2 Compliant Commercial Systems
==========

Orbeline

Provider(s): Orbeline
Contact Information: PostModern Computing Technologies, Inc.;
phone (415) 967-6169
Availability: licensed
Cost: Unknown
Platforms (Hdw &/or OS): Solaris1, Solaris2, OSF/1
Software dependencies: None
Version: ?
Source of Comparison:
Orbeline Reference Manual and Users Guide, 1994.

Component Coupling Model: Autonomous
Communication paradigms: RPC
Standards Adherence: CORBA 1.2
Multi-Protocol: No
Interface Compiler: Yes
Languages Supported: C, C++
Multi-Language Support: No
Dynamic Interfaces: Client-side
Location Transparency: Yes
Object Granularity: Local, Non-local
First Class Contained objects: No
Orthogonality: Yes
Combined Client+Server: Yes
Fault-Tolerance: Connection Loss Detection, Non-Transparent
Server Replacement
Exceptions: Yes
Authorization Control: Yes
Request Priorities: No
Receipt Acknowledgement: No
Futures: No
Threadable: Yes
Non-blocking substrate: No
Separate Marshaling: No
Versioning: No
Registration: Dynamic

Iona:

Provider(s): Iona
Contact Information: http://www.inona.ie
Availability: licensed
Cost: $5000
Platforms (Hdw &/or OS): Solaris1, Solaris2, NT, HPUX, IRIX
Software dependencies: None
Version: 1.2
Source of Comparison: ?

Component Coupling Model: Autonomous
Communication paradigms: RPC
Standards Adherence: CORBA 1.2
Multi-Protocol: No
Interface Compiler: Yes
Languages Supported: C, C++
Multi-Language Support: No
Dynamic Interfaces: Client-side
Location Transparency: Yes
Object Granularity: Local, Non-local
First Class Contained objects: No
Orthogonality: Yes
Combined Client+Server: Yes
Fault-Tolerance: Connection Loss Detection
Exceptions: Yes
Authorization Control: Yes
Request Priorities: No
Receipt Acknowledgement: No
Futures: No
Threadable: Yes
Non-blocking substrate: No
Separate Marshaling: No
Versioning: No
Registration: Dynamic

==========
SECTION III: Other Commercial Systems
==========

DCE:

Provider(s): OSF and various vendors
Contact Information: ?
Availability: Licensed
Cost: Vendor Specific
Platforms (Hdw &/or OS): Solaris1, Solaris2, AIX, ULTRIX, OSF/1,
HPUX, IRIX, NT
Software dependencies: None
Version: 1.1
Source of Comparison: OSF DCE Application Development Guide, Revision 1.0

Component Coupling Model: Autonomous
Communication paradigms: RPC
Standards Adherence: DCE
Multi-Protocol: No
Interface Compiler: Yes
Languages Supported: C, C++
Multi-Language Support: No
Dynamic Interfaces: No
Location Transparency: Yes
Object Granularity: Non-local
First Class Contained objects: No
Orthogonality: Yes
Combined Client+Server: Yes
Fault-Tolerance: Connection Loss Detection
Exceptions: Yes

Authorization Control: Yes
Request Priorities: No
Receipt Acknowledgement: No
Futures: Yes
Threadable: Yes
Non-blocking substrate: No
Separate Marshaling: No
Versioning: Yes
Registration: Dynamic

OLE2 (COM):

Provider(s): Microsoft
Contact Information: Microsoft
Availability: Bundled with Windows
Cost: N.A.
Platforms (Hdw &/or OS): Windows 3.1, Windows 95.
Software dependencies: None
Version: 2.1
Source of Comparison: OLE 2 Programmer's Reference, Volumes 1 and 2

Component Coupling Model: Autonomous
Communication paradigms: RPC
Standards Adherence: OLE2, DCE?
Multi-Protocol: No
Interface Compiler: Yes
Languages Supported: C, C++
Multi-Language Support: No
Dynamic Interfaces: Yes
Location Transparency: Yes
Object Granularity: Local
First Class Contained objects: Yes
Orthogonality: Unknown
Combined Client+Server: Unknown
Fault-Tolerance: Unknown
Exceptions: Yes
Authorization Control: Yes
Request Priorities: No
Receipt Acknowledgement: No
Futures: No
Threadable: Unknown
Non-blocking substrate: No
Separate Marshaling: No
Versioning: No
Registration: Static

OpenDoc ((D)SOM):

Provider(s): CLI
Contact Information: ftp://cil.org
Availability: Licensed
Cost: Unknown
Platforms (Hdw &/or OS): Macintosh
Software dependencies: None
Version: ?
Source of Comparison: SOMobjects Developer Toolkit Technical
Overview, version 2.0, November 1993.

Component Coupling Model: Autonomous
Communication paradigms: RPC
Standards Adherence: OpenDoc, CORBA 1.2
Multi-Protocol: No
Interface Compiler: Yes
Languages Supported: C, C++

Multi-Language Support: No
Dynamic Interfaces: Client-side
Location Transparency: Yes
Object Granularity: Local, Non-local
First Class Contained objects: Unknown
Orthogonality: Unknown
Combined Client+Server: Unknown
Fault-Tolerance: Unknown
Exceptions: Yes
Authorization Control: Yes
Request Priorities: No
Receipt Acknowledgement: No
Futures: Unknown
Threadable: Unknown
Non-blocking substrate: No
Separate Marshaling: No
Versioning: Yes
Registration: Dynamic

ISIS:

Provider(s): Isis Distributed Systems Inc.
Contact Information: Isis Distributed Systems Inc.; phone: 607-272-6327
Availability: licenced
Cost: Unknown
Platforms (Hdw &/or OS): Solaris1, Solaris2
Software dependencies: None
Version: 3.0
Source of Comparison: Isis Version 3.0 Reference Manual

Component Coupling Model: Autonomous
Communication paradigms: RPC
Standards Adherence: None
Multi-Protocol: No
Interface Compiler: No
Languages Supported: C, C++, Fortran, Lisp
Multi-Language Support: No
Dynamic Interfaces: Unknown
Location Transparency: Yes
Object Granularity: Non-local
First Class Contained objects: No
Orthogonality: Yes
Combined Client+Server: Yes
Fault-Tolerance: Connection Loss Detection, Transparent
Server Replacement
Exceptions: Yes
Authorization Control: No
Request Priorities: No
Receipt Acknowledgement: No
Futures: Yes
Threadable: Yes
Non-blocking substrate: No
Separate Marshaling: Yes
Versioning: No
Registration: Dynamic

==========
SECTION IV: Aggregate Groups of Related Systems
==========

Integrated Distributed Computations (E.g. Polylith):

Provider(s): University of Maryland
Contact Information: purtilo@cs.umass.edu

Availability: Public
Cost: free
Platforms (Hdw &/or OS): Solaris1
Software dependencies: None
Version: 2.1
Source of Comparison: Polylith 2.1 Distribution Documentation

Component Coupling Model: Tightly Integrated
Communication paradigms: RPC, Message
Standards Adherence: None
Multi-Protocol: No
Interface Compiler: Yes
Languages Supported: C, C++, Ada
Multi-Language Support: No
Dynamic Interfaces: No
Location Transparency: Yes
Object Granularity: Non-local
First Class Contained objects: No
Orthogonality: No
Combined Client+Server: Yes
Fault-Tolerance: Connection Loss Detection
Exceptions: No
Authorization Control: No
Request Priorities: No
Receipt Acknowledgement: No
Futures: Yes
Threadable: No
Non-blocking substrate: No
Separate Marshaling: No
Versioning: No
Registration: static

Message Broadcasters (E.g. ToolTalk):

Provider(s): Sun
Contact Information: SunSoft
Availability: licensed
Cost: Unknown
Platforms (Hdw &/or OS): Solaris1, Solaris2, HPUX.
Software dependencies: None
Version: 1.1.1
Source of Comparison: The ToolTalk Service, from SunSoft.

Component Coupling Model: Autonomous
Communication paradigms: Broadcast
Standards Adherence: Tooltalk
Multi-Protocol: No
Interface Compiler: N.A.
Languages Supported: C, C++
Multi-Language Support: Yes
Dynamic Interfaces: Yes
Location Transparency: Yes
Object Granularity: N.A.
First Class Contained objects: N.A.
Orthogonality: Yes
Combined Client+Server: Yes
Fault-Tolerance: Connection Loss Detection, Transparent
Server Replacement
Exceptions: No
Authorization Control: No
Request Priorities: Yes
Receipt Acknowledgement: No
Futures: N.A.
Threadable: Yes
Non-blocking substrate: No

Separate Marshaling: No
Versioning: Yes
Registration: Dynamic

Interoperability Working Group

ARPA/SISTO Environments Meeting
St. Louis, MO.
20-23 September 1994

Working Group Participants

Dennis Heimbigner, Chair (CU)
Lolo Penedo (TRW)
John Arnold (Bull)
Kurt Wallnau (SEI)
Ellis Horowitz (USC)
Mark Moriconi (SRI)

MOTIVATING PROBLEM

- Customer's goals for software environments:
  - Move from a custom-built, closed, programming environment to one that is
    1. *Rapidly assembled* using COTS/GOTS/ROTS components.
    2. *Tailorable* to mission-specific needs.
    3. *Evolvable* to keep pace with emerging environment technologies.
    4. *Scalable* to support large projects.
- Barrier to achieving the customer's goals:
  - Components often make incompatible assumptions about their operating context.

    ⇒ Components cannot interoperate.

SCOPE

- In-scope topics:
  - Evaluation of interoperability mechanisms.
  - Tailoring to specific domains.
  - Enhancement to existing mechanisms.
- Out-of-scope topics:
  - Development of new interoperability mechanisms that compete directly with commercial systems.
  - Specific tools or tool interfaces.
    - This is the purvue of specific working groups (e.g., process, object management).

RELATIONSHIP TO PROGRAM OBJECTIVES/STRATEGY

- Serve as a conduit for the principaled introduction of general interoperability technology into the software environments community.

EVIDENCE OF NEED

- CASE tool vendors have proven resistant to opening their tools.
  - Reason: loss of market advantage.
  - Failure examples: AD-CYCLE, PCTE, (and TI+Microsoft IEF effort?)
- Inability to combine the functionality of existing tools.
  - The "who's in charge?" problem: Every tool thinks it is in charge and all other tools must be subordinate.
  - Many tools can only be accessed through their user interface.
- Scalability of architectures.
  - The "PC mentality reigns in CASE tool market.
    ⇒ CASE tools typically target individual programmers.
- Environments require all-or-nothing buy-in.
  - Incremental insertion of pieces of the environment is often impossible.
- Difficulty in tailoring environments to specific domains.

TRENDS

KEY THRUSTS	TODAY	+ 2 YEARS	+4 YEARS	IMPACT/METRICS
Environments are collections of interoperating large-grain objects	– Vendor coalitions – Event-level integration (X3H6) – File export/import – CORBA ground-swell	– Limited availablity of CORBA-based components – Interoperable ORBS (CORBA2) – ADL representations of "standard" environment architectures – Prototypes of (semi) automated "glue" generation for architecture based composition	– Wide-spread availability of CORBA based components – Successor to CORBA emerges (CORBA++)	– Marketplace for environment components – "Rapid" assembly of environments from third party component vendors
Component interoperability is essential for new software environment requirements such as Collaboration and Process	– Vendor-specific, non-interoperating, point solutions – Event-based approaches in research prototypes – Separate mechanisms (MBONE) for multi-media – Sparse, ad-hoc examples of interoperability of process engines	– Prototype CORBA +MBONE collaborative applications – Prototype interoperation of process engines at multiple sites	– Real-time, multi-media extensions to CORBA – Principaled interoperability between process engines	– Marketplace for process components – Flexible combinations of autonomous processses – High-performance, widely available collaborative applications
Existing mechanisms must be evaluated against above concepts (quantifable/qualifiable)	– Trade-Journal driven push for use of CORBA as standard – Throughput measures available, but no standard benchmarks – Insufficient experience to reveal functional deficiencies of CORBA	– Ad-hoc performance measures of ORB to ORB interoperability – Development of prototype benchmark standards – Functional deficiencies revealed by initial experience	– General availability and use of standard benchmark data – Prototypes of extensions to resolve discovered deficiencies	– Informed selection of mechanisms and reduced technical risk – Improved performance and functionality for interoperability mechanisms

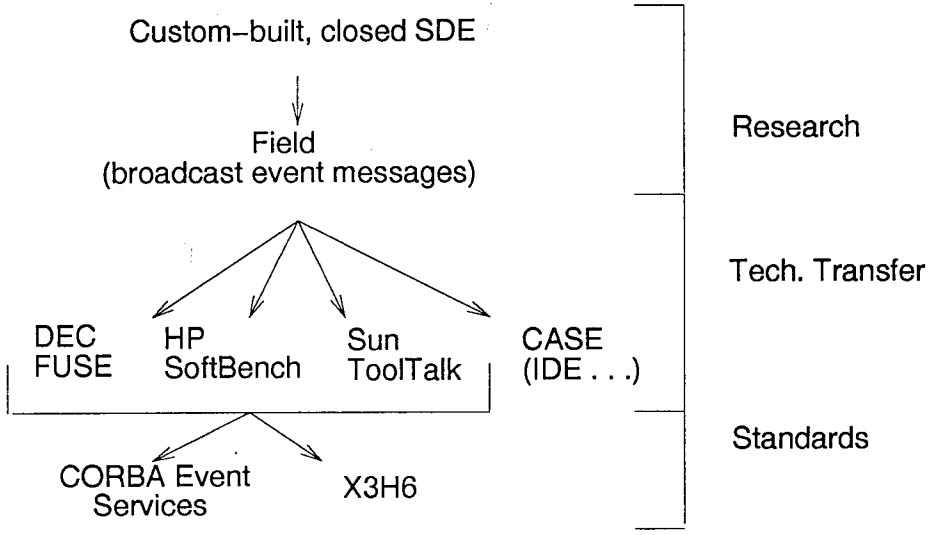
– Note: The term CORBA generically includes other mechanisms (e.g. OLE).

OBJECTIVES

- Move the community from production of interoperability technology to the consumption and evaluation of existing technology.
- Evaluate the existing major standards to determine their limitations with respect to the requirements of the environments community.
- Define criteria and provide guidance in selecting interoperability technology.
- As needed, develop prototype solutions to address discovered deficiencies in existing technologies.

ACCOMPLISHMENTS

- Steve Reiss's Field broadcast message system must be considered a major sucess coming out of this community.
  - o Field provided the first step in moving from closed, custom software environments to open, tailorable environments.
  - o Field has been picked up used commercially by all major software producers and CASE vendors.
  - o Secondarily, Field has influenced the X3H6 and CORBA Event Services Standards.



ACCOMPLISHMENTS (cont.)

- Successful demonstrations of environments integrated by more than just the file system.
  - Coarse-grain event integration (e.g. Field, Arcadia, Conversation Builder).
  - Component-object environments (e.g. Arcadia, Marvel).
- The Arcadia Q System.
  - Allowed Arcadia to obtain experience with distributed-component based environments
    - This experience will significantly reduce the costs in moving to CORBA-based environments.
  - Provides superior Ada support for remote-procedure call.
  - Tech transfer to STRICOMM for the STOW project.
- Conversation Builder + Marvel
  - Tech transfer to Bull Corp. to support collaborative meeting technology.

ROAD-MAP

- Years 1 and 2
  - Re-host of major environment projects (e.g. Arcadia) onto CORBA-based mechanisms.
  - Initial experiments to combine process and collaboration support with CORBA.
  - Initial performance and functionality benchmarks for measuring ORBS.
- Years 3 and 4
  - General availability of combined process and collaboration support with ORBS.
  - Standardized performance benchmarks for measuring ORB performance and ORB to ORB performance.
  - Prototyped solutions to ORB functional deficiencies.

Architecture Bibliography
February 1995

References

[1] Software Bus - Rationale. Technical report, ESF-SwB Sub-Project, June 1990.

[2] Application Portability Profile (APP). Technical Report NIST Special Publication 500-187, OSE/1 Version 1.0, April 1991.

[3] RAPIDE 1.0 Executable Language Reference Manual. Technical report, Stanford University, August 17 1992.

[4] Reference Model for Frameworks of Software Engineering Environments. Technical Report NIST Special Publication 500-211 and ECMA/TC33 Technical Report TR/55, 3rd Edition, August 1993.

[5] H. Achkar et al. Software Factory Reference Framework. Technical Report Report of the Reference Model WG, ESF, October 1990.

[6] R. Adomeit, W. Deiters, et al. K/2R: A Kernel for the ESF Software Factory Support Environment. In *2nd International Conference on Systems Integration 92*, Morristown, NJ, June 1992.

[7] R. Adomeit and B. Holtkamp. ESF Factory Support Environment: Architectural Refinements and Alternatives. Technical report, University of Dortmund.

[8] B. Beach. Connecting Software Components with Deaclarative Glue. In *ACM 0-89791-504-6*, 1992.

[9] I. Z. Ben-Shaul and G. E. Kaiser. A Paradigm for Decentralized Process Modeling and its REalization in the Oz Environment. In *16th International Conference on Software Engineering*, pages 179–189.

[10] T. Bingen, R. Foulkes, and L. Morgan. Data and Control Integration in a Software Factory: in Software Bus. Technical report, Sema Group - Brussels, Glasgow, Paris.

[11] A. Brown, D. Carney, P. Oberndorf, and M. Zelkowitz editors. Reference Model for Project Support Environments. Technical Report Technical Report CMU/SEI-93-TR-23, ESC-TR-93-199, U.S. Navy NGCR Program, also published as NIST Special Publication 500-213, November 1993.

[12] A. Brown and M. H. Penedo. An Annotated Bibliography on Integration in Software Engineering Environments. In *ACM Software Engineering Notes*, July 1992.

[13] B.W. Boehm and W. L. Scherlis. Megaprogramming. In *Proceedings of the DARPA Software Technology Conference*, April 1992.

[14] Purtilo C. Hofmeister, J. Atlee. *Writing Distributed Programs in Polylith.* University of Maryland, November 1990.

[15] M. Cagan. An Encapsulator Tutorial. SoftBench Technical Note Series SESD-89-15 Revision 1.1, Hewlett-Packard, Fort Collins, Colorado, 80525, August 1989.

[16] M. Cagan. The McCabe Encapsulation. SoftBench Technical Note Series SESD-89-18 Revision 1.0, Hewlett-Packard, Software Engineering Systems Division, 3404 E. Harmony Road, Fort Collins, Colorado 80525, August 1989.

[17] M. Cagan. The HP SoftBench Environment: An Architecture for a New Generation of Software Tools. *Hewlett-Packard Journal*, June 1990.

[18] C. Chen, E. L. White, and J. M. Purtilo. A Packager for Multicast Software in Distributed Systems. Technical report, University of Maryland, 1992.

[19] H. Davidson. Encapsulator: The Plug-In Compatibility Tool for SoftBench. SoftBench Technical Note Series SESD-89-11 Revision 1.1, Hewlett-Packard, Software Engineering Systems Division, 3404 E. Harmony Road, Fort Collins, Colorado 80525, June 1989.

[20] Christer Fernstrom. Process Weaver: Adding Process Support to UNIX. In *Proceedings of the Second International Conference on the Software Process*, Berlin, Germany, February 1993.

[21] R. Foulkes. The ESF Software Bus. Technical report, Yard Ltd., UK - Glasgow, 1990.

[22] R. Foulkes, O. Nanlot, T. Bingen, and C. Ginn. The Software Bus - An Overview. Technical report, Sema Group Belgium, September 5, 1991 1991.

[23] D. Garlan, R. Allen, and J. Ockerbloom. Exploiting Style in Architecture Design Environments. In *2nd ACM SIGSOFT Symposium on Foundations of Software Engineering*, New Orleans, Louisiana, December 1994.

[24] D. Garlan and E. Ilias. Low-cost, Adaptable Tool Integration Policies, for Integrated Environments. In *4th ACM SIGSOFT Symposium on Software Development Environments*, Irvine, CA, December 1990.

[25] D. Garlan and M. Shaw. An Introduction to Software Architecture. Technical Report CMU/SEI-93-TR-33, Carnegie Mellon University, 1993. also in Ambriola, V; and Tortora, G, Advances in Software Engineering and Knowledge Engineering, Vol. 1, Singapore, World Scientific Publishing.

[26] Michael Gera, Bruno Hirsch, Bernard Holtkamp, Jean-Pierre Moularde, Graham Samuel, and Herbert Weber. CoRe, A Conceptual Reference Model for Software Factories. Technical report, Eureka Software Factory, November 25 1992.

[27] M. Gorlick. Cricket: A Domain-Based Message Bus for Tool Integration. Technical report, The Aerospace Corporation.

[28] M. Gorlick and R. Razouk. Using Weaves for Software Construction and Analysis. In *13th International Conference on Software Engineering*, Austin, Texas, May 1991.

[29] D. Heimbigner. The Process Wall: A Process State Server Approach to Process Programming. In *Proceedings of the 5th ACM SIGSOFT Symposium on Software Developement Environemnts - ACM Software Engineering Notes*, December 1992.

[30] D. H. Heimbigner. ARPC: An Augmented Remote Procedure Call System. Technical Report CU-Arcadia-100-92, Department of Computer Science, University of Colorado, October 19 1992.

[31] B. Holtkamp. Service Component Reference Model. Technical Report ESF Internal paper, University of Dortmund, march 1991.

[32] B. Holtkamp. MUSE - A Framework for Decentralized Software Development Environments. Technical report, University of Dortmund, 1992.

[33] B. Holtkamp. MUSE Interoperation Support. Technical report, Fraunhofer Institue for Software and Systems Technology, 1992.

[34] B. Holtkamp and F. Schuelke. The Software Bus - Communication Aspects. Technical report, University of Dortmund, March 18 1991.

[35] B. T. Jenings, Jr. The HP SoftBench Message Model: Concepts and conventions used by the Hp SoftBench Tools. SoftBench Technical Note Series SESD-89-21 Revision 1.2, Hewlett-Packard, Fort Collins, Colorado 80525, September 1989.

[36] R. Kadia. Issues Encountered in Building a Flexible Software Development Environment - Lessons from the Arcadia Project. In *Proceedings of the 5th ACM SIGSOFT Symposium on Software Development Environments - ACM Software Engineering Notes*, Virginia, USA, December 1992.

[37] A. Karrer, M. H. Penedo, and C. Shu. A Survey of Software Engineering Environment Architecture Approaches. Technical Report Arcadia-TRW-93-007, TRW, Redondo Beach, CA, 1990, November 1993.

[38] S. A. Kramer. SoftBench DM Message Integration Requirements. SoftBench Tecnhical Note Series SESD-89-20 Revision 1.2, Hewlett-Packard, Fort Collins, Colorado, August 1989.

[39] B. Liskov. Distributed Programming in Argus. *Communications of ACM*, pages 300–312, March 1988.

[40] J. Magee, J. Kramer, and M. Sloman. Constructing Distributed Systems in Conic. *IEEE Transactions on Software Engineering*, June 1989.

[41] M. Maybee, D. Heimbigner, D. Levine, and L. Osterweil. Q: A Multi-lingual Interprocess Communications System for Software Environment Implementation. Technical Report CU-CS-92, Department of Computer Science, University of Colorado, 1992.

[42] L. Morgan. *The Software Bus - User Requirements.* Sema Group Belgium, 4.2 edition, September 1992.

[43] L. Morgan. *The Software Bus - Reference Manual.* Sema Group Belgium, 4.0 edition, March 1993.

[44] L. Morgan. *The Software Bus - User's Guide.* Sema Group Belgium, 4.0 edition, March 1993.

[45] M. Moriconi and X. Qian. Correctness and Composition of Software Architectures. In *2nd ACM SIGSOFT Symposium on Foundations of Software Engineering*, New Orleans, Louisiana, December 1994.

[46] B. A. Nejmeh. Characteristics of Integrable Software Tools. Technical Report Integ S/W Tools-89036-N, Software Productivity Consortium, May 1989. Version 1.0.

[47] OMG. Object Services Architecture. Technical Report OMG Document Number 92.8.4, Object Management Group, Object Management Group, Inc., Headquarters, 492 Old Connecticut Path, Framingham, Ma. 01701, August 1992.

[48] OMG. The Common Object Request Broker: Architecture and Specification. Technical Report OMG Document Number 91.12.1 Revision 1.1, Object Management Group, Object Management Group, Inc., Headquarters, 492 Old Connecticut Path, Framingham, Ma. 01701, 1992.

[49] OMG. Object Request Broker Architecture. Technical Report OMG TC Document 93.7.2, Object Management Group, Object Management Group, Inc., Headquarters, 492 Old Connecticut Path, Framingham, Ma. 01701, 1993.

[50] M. H. Penedo. Different perspectives on integration. In *Proceedings of Process Sensitive SEE Architecture Workshop*, Boulder, CO, September 1992.

[51] M. H. Penedo. Towards understanding Software Engineering Environments. In *Proceedings of TRW Conference on Integrated Computer-Aided Software Engineering*, California, November 1993. also in TRW Technical Report IMPSEE-TRW-93-003.

[52] M. H. Penedo and D. Berry. The Use of a Module Interconnection Language in the SARA System Design Methodology. In *Proceedings of the 4th International Conference on Software Engineering*, Munich, Germany, September 1979.

[53] M. H. Penedo and W.E. Riddle. Process-sensitive Software Engineering Environment Architectures - Summary Report. In *ACM Software Engineering Notes*, July 1993.

[54] M.H. Penedo, E. Ploedereder, and I. Thomas. Object Management Issues for Software Engineering Environments - Workshop Report. In *Proceedings of the 3rd ACM Software Engineering Symposium on Software Development Environments*, Boston, November 1988.

[55] M.H. Penedo and C. Shu. Acquiring Experiences with the Modeling and Implementation of the Project Life-cycle Process - the PMDB work. *IEE and British Computer Society Software Engineering Journal*, September 1991.

[56] D. E. Perry and G. E. Kaiser. Models of software development environments. In *Tenth International Conference on Software Engineering*, pages 60–66, April 1988.

[57] Dewayne E. Perry and Alexander L. Wolf. Software Architecture. Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey, September 1989. Revised January 1991.

[58] Dewayne E. Perry and Alexander L. Wolf. Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*, (17,4):40–52, October 1992.

[59] J. Purtilo. Polylith: An Environment to Support Management of Tool Interfaces. In *ACM SIGPLAN Symposium on Language Issues in Programming Environments*, July 1985.

[60] J. Purtilo. The Polylith Software Bus. Technical Report TR-2469, University of Maryland, April 1991.

[61] J. M. Purtilo, R. T. Snodgrass, and A. L. Wolf. Software Bus Organization: Reference Model and Comparison of Two Existing Systems. Technical Report Technical Note No. 8, DARPA Module Interconnection Formalism Working Group, November 1991.

[62] S.C. Bandinelli and A. Fuggetta and C. Ghezzi. Software process model evolution in the spade environment.

[63] F. Schuelke. The MUSE System. Technical report, Eureka Software Factory, April 25 1991.

[64] M. Shaw. Larger Scale Systems Require Higher-Level Abstractions. In *Proceedings of the Fifth International Workshop on Software Specification and Design*. IEEE Computer Society, 1989.

[65] M. Shaw. Heterogeneous Design Idioms for Software Architecture. In *Proceedings of the Sixth International Workshop on Software Specification and Design*, Como, Italy, October 25-26 1991. IEEE Computer Society.

[66] C. Shu and M. H. Penedo. SEE Software Bus Survey. Technical Report IMPSEE-TRW-93-008, TRW, Redondo Beach, CA, December 1993.

[67] S. Sutton, Jr. and M. H. Penedo. Process-based SEE Architectures Session Report. In *Proc. of the 7th International Software Process Workshop*, California, October 1991.

[68] G. Tatge. HP SoftBench. In *FedCASE '89*, 1989.

[69] R.N. Taylor et al. Foundations for the Arcadia Environment Architecture. In *Proceedings of the 3rd ACM Software Engineering Symposium on Software Development Environments*, Boston, November 1988.

[70] UniDo. Kernel/2 Definition. Technical Report ESF, University of Dortmund, September 1991.

[71] J. Veijalainen, F. Eliassen, and B. Holtkamp. *The S-transaction Model.* Morgan Kaufmann, July 12 1991.

[72] K. Wallnau and P. Feiler. Tool Integration and Environment Architectures. Technical Report CMU/SEI-91-TR-11, Carnegie Mellon University, May 1991.

[73] J. C. Wileden, A. L. Wolf, W. R. Rosenblatt, and P. L. Tarr. Specification Level Interoperability. In *Proceedings of the 12th International Conference on Software Engineering*, Nice, France, March 1990.

11. PSEE Tutorial: Trends in the Construction of Next Generation Software Engineering Environments, by M. Penedo.

[Included as an attachment]